DÉDICACES

A mes très chers parents

Vous m'avez soutenu tout au long de mes études et vous m'avez donné un modèle magnifique de labeur et de persévérance. C'est grâce à votre affection et vos sacrifices que j'ai réussi ce travail. Que vous trouvez ici le témoignage de ma reconnaissance, amour et respect, que Dieu tout puissant vous bénisse et vous procure santé et longue vie.

A mes chers frères et sœurs

Je tiens à vous remercier pour votre soutien exemplaire et les efforts que vous n'avez jamais cessé de consentir. Je vous rends hommage par ce modeste travail, en guise de ma gratitude éternelle. Je vous souhaite la joie, la prospérité et la réussite que vous méritez.

A mes amis

J'espère avoir répondu aux espoirs que vous avez fondés en moi.

Veuillez accepter l'expression de mon amour, ma profonde gratitude pour votre amitié, compréhension et encouragements. Que Dieu vous protège et vous procure joie et bonheur et que notre amitié reste à jamais.

A tous ceux ou celles qui me sont chers et que j'ai omis involontairement de citer.

A tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail Puisse l'éternel nous guide vers le succès et la gloire.

Sabri BAHRINI

REMERCIEMENTS

Au terme de ce stage de fin d'étude, je consacre ces mots en signe de reconnaissance, à toutes les personnes qui ont contribué à l'élaboration de ce travail. J'adresse mes remerciements les plus sincères à :

Mlle. Nadia BEN YOUSSEF

Enseignante à l'ISI, pour son encadrement, sa disponibilité et ses conseils fructueux prodigués le long du projet.

M. Chihebeddine ZRIBI

Mon responsable de stage à la société LYSI ENGENEERING*, pour son soutien technique, ses critiques constructives, et l'aide qu'il m'a accordé.*

Tous nos enseignants à l'ISI

Pour leur formation tout au long de mes études et à tous ceux qui m'ont fourni les moyens techniques pour accomplir ce travail dans les bonnes conditions.

ET

expérience avec moi, et qui ont témoigné un soutien sans faille et des encouragements qui m'ont permis de mener à bien ce travail.

Que tous ceux qui ont contribué de près ou de loin à ce projet y trouvent la reconnaissance de leurs efforts et le fruit de leur labeur.

Je tiens également à remercier les membres de jury de m'avoir honoré en acceptant d'évaluer mon travail.

Sabri BAHRINI

Table des matières

INTRODUCTION GÉNÉRALE **1**

I CADRE GÉNÉRAL DU PROJET **3**
Introduction . . . 4
1 Présentation de l'organisme d'accueil . . . 4
1.1 Le marché . . . 4
1.2 Service . . . 4
1.3 Perspectives de l'entreprise . . . 5
2 Présentation du projet . . . 5
2.1 Contexte du projet . . . 5
2.2 Problématique . . . 6
2.3 Objectif du projet . . . 6
3 Étude et critique de l'existant . . . 7
4 Méthodologie adoptée . . . 7
4.1 Présentation du langage de modélisation UML . . . 8
4.2 Processus de développement logiciel . . . 8
4.3 Présentation d'AUP (Agile Unified Process) . . . 8
5 Chronogramme de réalisation . . . 9
Conclusion . . . 9

II DÉFINITION ET SPÉCIFICATION DES BESOINS **10**
Introduction . . . 11
1 Analyse des besoins . . . 11
1.1 Besoins fonctionnels . . . 11
1.2 Besoins non fonctionnels . . . 12
2 Solution proposée . . . 12
2.1 Identification des acteurs . . . 12
2.2 Identification des cas d'utilisations . . . 13
2.2.1 Diagramme de cas d'utilisation général . . . 13
2.2.2 Cas d'utilisation « Compiler structure XML » . . . 16
2.2.3 Cas d'utilisation « Créer application » . . . 17
2.2.4 Cas d'utilisation « Modifier application » . . . 18
2.2.5 Cas d'utilisation « Construire interface » . . . 19
2.2.6 Cas d'utilisation « Gérer fichiers de configuration » . . . 20
2.2.7 Cas d'utilisation « Gérer données » . . . 21
2.2.8 Cas d'utilisation « S'authentifier » . . . 22
3 Maquette IHM . . . 23
Conclusion . . . 23

III CONCEPTION **24**
Introduction . . . 25
1 Vues Dynamiques . . . 25
1.1 Diagrammes de séquences . . . 25
1.1.1 S'authentifier . . . 26
1.1.2 Compiler structure XML . . . 27
1.1.3 Créer Application . . . 28
1.1.4 Modifier application . . . 29
1.1.5 Construire interface . . . 30
1.1.6 Gérer fichiers de configuration . . . 31
1.2 Diagrammes d'activités . . . 32
1.2.1 Diagramme d'activités général . . . 32
1.2.2 Diagramme d'activités « Compiler structure XML » . . . 32
2 Vues Statique . . . 33
2.1 Diagramme de classes . . . 34
2.2 Diagramme de déploiement . . . 35
Conclusion . . . 36

IV RÉALISATION **37**
Introduction . . . 38
1 Choix technologique . . . 38
1.1 PHP 5 . . . 38
1.2 HTML 5 . . . 38
1.3 JSON . . . 39
1.4 JQUERY . . . 39
1.5 CSS3 . . . 39
2 Choix architectural . . . 39
2.1 Le modèle MVC . . . 40
2.1.1 Le modèle . . . 40
2.1.2 La vue . . . 40
2.1.3 Le contrôleur . . . 41
2.2 Architecture 3-tiers . . . 41
3 Environnement matériels . . . 42
3.1 Machines utilisés . . . 42
3.2 Architecture système . . . 42
3.2.1 Serveur MYSQL . . . 43
3.2.2 Serveur HTTP Apache . . . 43
4 Environnement logiciel . . . 43
4.1 MySQL Workbench . . . 43
4.2 Filezilla . . . 43
4.3 L'environnement de développement NetBeans . . . 44
5 Scénarios d'utilisation . . . 44
5.1 Authentification développeur . . . 44
5.2 Chargement des fichiers dans le serveur web . . . 45
5.3 Génération d'une nouvelle application . . . 46
5.4 Authentification d'utilisateur simple . . . 46
5.5 Accès à l'interface du la plateforme . . . 47
6 Définition de système objet relationnel (S.O.R) . . . 48
6.1 Le Modèle S.O.R . . . 48
6.1.1 La DAL (Data Access Layer) . . . 49

6.1.2 Le DAO (Data Access Object) 49
6.1.3 La couche métier de S.O.R 49
6.2 Le contrôleur S.O.R 50
6.3 La vue S.O.R 50
6.3.1 Menu 51
6.3.2 Vue 52
6.3.3 Formulaire 52
6.3.4 Dialogue 53
7 Définition du compilateur de S.O.R 54
Conclusion 55

CONCLUSION ET PERSPECTIVES **56**

Table des figures

I.1 Simplification du modèle de l'infrastructure informatique d'une entreprise . . 4
I.2 Mutualisation des services 5
I.3 Chronogramme 9

II.1 Diagramme de cas d'utilisation général 14
II.2 Diagramme de cas d'utilisation « Compiler structure XML » 16
II.3 Diagramme de cas d'utilisation « Créer application » 17
II.4 Diagramme de cas d'utilisation « Modifier application » 18
II.5 Diagramme de cas d'utilisation « Construire interface » 19
II.6 Diagramme de cas d'utilisation « Gérer fichiers de configuration » 20
II.7 Diagramme de cas d'utilisation « Gérer données » 21
II.8 Diagramme de cas d'utilisation « S'authentifier » 22
II.9 Maquette IHM par défaut 23

III.1 Diagramme de séquence « S'authentifier » 26
III.2 Diagramme de séquence « Compiler structure XML » 27
III.3 Diagramme de séquence « Créer Application » 28
III.4 Diagramme de séquence « Modifier application » 29
III.5 Diagramme de séquence « Construire interface » 30
III.6 Diagramme de séquence « Gérer fichiers de configuration » 31
III.8 Diagramme d'activités « Compiler structure XML » 32
III.7 Diagramme d'activités général 33
III.9 Diagramme de classes 34
III.10 Diagramme déploiement 36

IV.1 Le modèle MVC 40
IV.2 Architecture 3-tiers 41
IV.3 Architecture logique du système 42
IV.4 Interface d'authentification développeur 45
IV.5 Interface (Filzilla) de communication avec le serveur web 46
IV.6 Interface d'authentification des utilisateurs 46
IV.7 Interface de message d'erreur d'authentification 47
IV.8 Interface de la plateforme générée par S.O.R 47
IV.9 Les différentes couches du système objet relationnel (S.O.R) 48
IV.10 Exemple d'un menu final d'une application gérer par S.O.R 51
IV.11 Exemple d'un menu secondaire 51
IV.12 Exemple d'une DataGrid 52
IV.13 Exemple d'un formulaire 52
IV.14 Exemple de dialogue 53
IV.15 Exemple de dialogue message 53
IV.16 Exemple de message de notification 53
IV.17 Compilateur S.O.R 54

Liste des tableaux

II.1 Description du cas d'utilisation « Compiler structure XML » 16
II.2 Description du cas d'utilisation « Créer application » 17
II.3 Description du cas d'utilisation « Modifier application » 18
II.4 Description du cas d'utilisation « Construire interface » 19
II.5 Description du cas d'utilisation « Gérer fichiers de configuration » 20
II.6 Description du cas d'utilisation « Gérer données » 21
II.7 Description du cas d'utilisation « S'authentifier » 22

Liste des acronymes

AUP Agile Unified Process
BD Base de données
CSS Cascading Style Sheets : feuilles de style en cascade
MVC Modèle-Vue-Contrôleur
ORM Mapping objet-relationnel
UML Unified Modeling Language
UP Unified Process
XML Extensible Markup Language
IHM Interactions homme-machine
TDD Test Driven Development
RUP Rational Unifies Process
XP eXtrema Programming
FTP File Transfer Protocol
FTPS File Transfer Protocol Secure
GNU GNU's Not UNIX
GNU GPL GNU General Public License
SOR Système objet relationnel
EDI environnement de développement intégré
CDDL Common Development and Distribution License
PHP Hypertext Preprocessor
HTML Hypertext Markup Language
SGBD Système de Gestion base de données
SFTP SSH File Transfer Protocol
DTD Document Type Definition

INTRODUCTION GÉNÉRALE

La relation d'objet est le rapport du sujet avec les objets qui constituent le monde dans lequel il est conçu. Ce monde est d'essence tout autant interne qu'externe. Ce qui rend la notion d'objet à la fois aussi vaste et ambiguë que difficile à définir et ambivalente. D'où les diverses approches et réflexions chez les théoriciens de cette notion.

Le modèle objet-relationnel est, en informatique, une manière de modéliser les informations contenues dans une base de données qui reprend le modèle relationnel en ajoutant quelques notions empruntées au modèle objet, venant combler les plus grosses lacunes du modèle relationnel. La technologie objet-relationnelle est née en 1992, elle est donc assez nouvelle sur le marché des SGBD, dominé depuis environ 1970 par les bases de données relationnelles sauf que la migration en objet relationnel est parfois lourde à mettre en place.

La solution était de créer une illusion d'une base de données orientée objet à partir d'une base de données relationnelle en associant une classe à une table, les attributs aux champs de la table. Cette démarche est assuré par le mapping objet-relationnel qui va permettre de faire un lien direct entre le monde objet et le monde relationnel. Ces ORM (Object/Relational Mapping) proposent de créer une couche entre une application et les données sous-jacentes en assurant la gérance de la persistance des données tout en gardant une architecture claire, concise et efficace.

Dans ce contexte, LYSI ENGENEERING a eu l'idée de développer une plateforme permettant de réduire les tâches répétitives de développement, à savoir l'utilisation des requêtes SQL, le développement des interfaces, la génération des interfaces et les bases de données depuis un fichier XML.

L'idée de notre travail répond à l'expression du besoin d'un environnement qui offre non seulement une modélisation visuelle pour la conception de données logique et la conception physique de données, mais aussi permet d'automatiser la mise en correspondance entre le modèle d'objet et le modèle de données.

Afin d'illustrer la démarche de notre travail, nous présentons dans ce qui suit l'organisation générale de notre rapport.

Tout d'abord nous allons présenter le cadre du projet ainsi la méthodologie adoptée pour finir avec l'étude et la critique de l'existant.

Le deuxième chapitre comporte l'analyse des besoins qui a permis de dégager les bases sur la quelle se pose notre application.

Dans un troisième chapitre et à partir de la description fonctionnelle obtenue lors de la phase de la spécification des besoins, nous avons décidé l'architecture de l'application pour finir par une conception décrite à l'aide d'une vue statique et dynamique.

Enfin, au cours du dernier chapitre, nous allons illustrer à travers un ensemble d'interfaces graphiques la phase de la réalisation du projet en passant par une description de l'environnement du travail.

Chapitre I

CADRE GÉNÉRAL DU PROJET

Introduction

Dans ce premier chapitre, nous allons commencer par présenter notre organisme d'accueil ensuite nous allons situer notre projet dans son cadre général suivi d'une description et critique de l'existant au sein de la société pour proposer enfin une solution pour laquelle nous avons opté.

1 Présentation de l'organisme d'accueil

LYSI ENGENEERING est une entreprise elle réalise des projets d'ingénierie de systèmes d'information, crée des annuaires et des portails web complets et consistants pour le secteur économique.

1.1 Le marché

Notre entreprise d'accueil vise les PMEs qui n'ont pas forcément les moyens d'acquérir une infrastructure logicielle complète. Elle propose également des prestations logicielles en mode SaaS.

1.2 Service

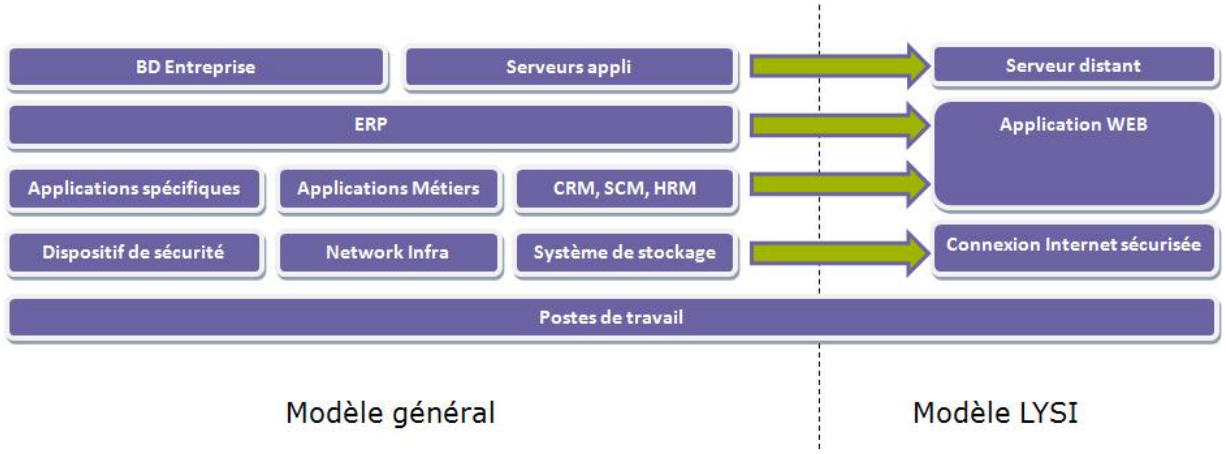

FIGURE I.1 – Simplification du modèle de l'infrastructure informatique d'une entreprise

La solution logicielle qu'elle propose offre :

- Un meilleur contrôle des charges techniques pour le client.
- Le prix par utilisateur englobe le coût des licences des logiciels, de la maintenance et de l'infrastructure.

– Coût moindre que l'acquisition des licences .
– Rapidité de déploiement.

Elle réalise aussi des sites et des portails web.

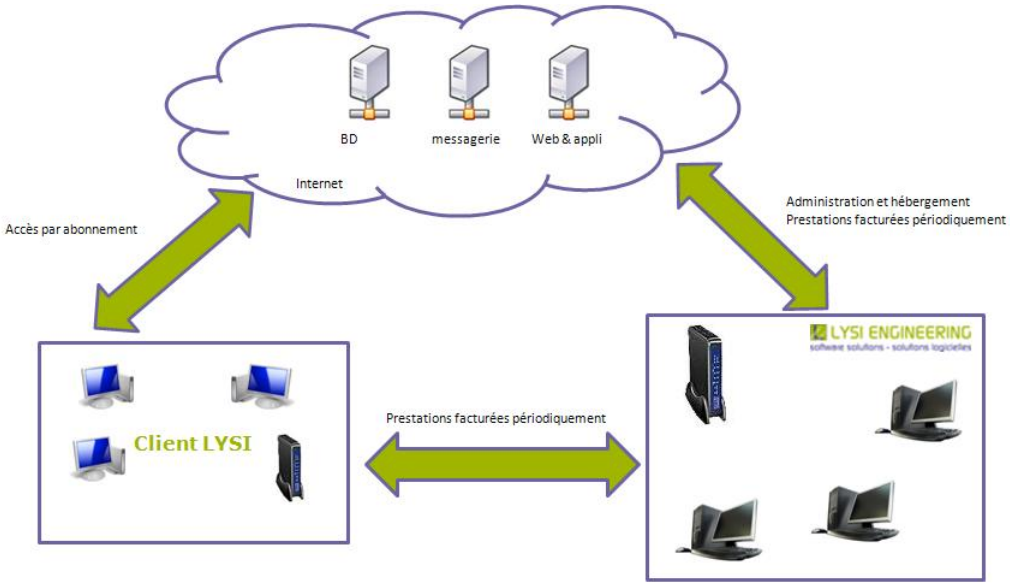

FIGURE I.2 – Mutualisation des services

1.3 Perspectives de l'entreprise

En fonction de la croissance du nombre des clients, de nouvelles dispositions peuvent être prises, une fois LYSI est autonome, elle souhaite créer une plateforme d'hébergement propre à elle.

2 Présentation du projet

Dans cette partie nous allons mettre le projet dans son cadre général, présenter son objectif ainsi que les problématiques qui en sont à l'origine.

2.1 Contexte du projet

LYSI ENGINEERING réalise des projets d'ingénierie de systèmes d'information, principalement destinées au monde professionnel. Elle souhaite améliorer son secteur de services et étendre son offre à un plus large public, afin d'être plus compétitive et plus rigide sur le marché. Notre projet de fin d'études vise à contribuer à l'étude et à la réalisation d'un ORM. C'est une technique de programmation faisant le lien entre le monde de la base de données et le monde de la programmation objet qui va nous faciliter énormément le travail de développement des applications web et minimiser son cout, à la fois attractive pour les

utilisateurs, puissante et facile à maintenir, tout en exploitant les compétences de la société dans le domaine.

2.2 Problématique

Le monde de développement des applications web présente plusieurs problématiques qui nous ont laissé penser à réaliser ce projet parmi lesquelles il y a :

- L'absence de la couche d'abstraction nécessaire pour lier le modèle aux ressources de l'application qui ce ne trouve pas dans les ORM existant.
- L'absence de génération automatique des bases de données.
- L'inutilisation des fichiers XML pour la création d'une base de données.
- L'absence de génération des Vues (Interfaces) pré-configurées.
- Le temps important d'exécution des requêtes et des scripts pour les différents ORM existants.
- L'utilisation de la programmation orientée objet avec une base de données relationnelle nécessite de convertir les données relationnelles en objets et vice-versa. Ceci conduit à programmer cette conversion pour chaque objet et donc à dupliquer énormément de code similaire.

2.3 Objectif du projet

Le développement d'une couche d'un modèle d'abstraction à la base de données nous permet de transformer une table en objet facilement manipulable via ses attributs.
D'où cette couche offre à ses utilisateurs :

- L'accès directe à la structure et les données d'une base de donnée.
- La génération automatique des interfaces (Vue), des classes (Contrôleur) et une base de donnée (Modèle) qui sont créer au début de la création de l'application web à laide des fichiers de configuration XML et un script de compilation écrit avec PHP5.
- La connexion avec les différents types de base de donnée (MySQL, PostgreSQL, Oracle, Sybase, SQL Server).
- La génération des fichiers des classes Business (Méthodes CRUD, surcharge des fonctions).

- La génération de la base de données (création et mise à jour de tables, création et mise à jour de déclencheurs).

3 Étude et critique de l'existant

L'utilisation des ORM lors du développement PHP représente de plus une nécessité sur le plan pratique, nous trouvons pas mal d'alternatives à savoir :

– Doctrine c'est un ORM pour PHP. Il s'agit de l'ORM par défaut du framework Symfony, il propose donc plusieurs fonctionnalités permettant de manipuler le modèle, mais aussi des fonctionnalités permettant d'interagir avec les données persistantes de l'application (accès en lecture de ces données, mais aussi insertion, mise à jour et suppression de celles-ci). Or les données persistantes traitées par ce ORM sont uniquement celles stockées dans des bases de données relationnelles, nécessite l'apprentissage d'un tout nouveau langage de programmation (Doctrine Query Language), et nécessite introspection de base de données et est donc légèrement plus lent que Propel et leur mise en cache ajoute une complexité considérable.[2]

– Propel c'est un ORM open-source pour PHP5, il nous permet d'accéder à nos base de données en utilisant un ensemble d'objets, en fournissant une API simple pour stocker et récupérer des données, Or il nécessite une étape de génération supplémentaire, à savoir la construction des classes de modèle, le code généré besoins reconstruit chaque fois que sa version ce changé, un paramètre est modifié, ou les changements de schéma, et certaines fonctionnalités utiles (comportement de version, les migrations de schéma) sont en état bêta.[11]

4 Méthodologie adoptée

Une méthodologie de génie logiciel décrit les différentes phases et leurs enchaînements dans le processus de développement d'une application informatique. Les plus récurrents dans ces méthodologies sont l'analyse de l'existant, la spécification, la conception, l'implémentation et l'évaluation. Une phase d'intégration est parfois explicitée lorsqu'il s'agit de combiner deux processus de développement relatifs à deux domaines différents.

Pour passer des besoins du Société, et des idées du projet, au code de l'application nous devons adopter une méthodologie de développement que nous allons respecter tout au long du projet afin de modéliser et développer notre application.

4.1 Présentation du langage de modélisation UML

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.

Nous avons opté pour UML parce-que c'est la méthode la plus adaptée à notre projet et aux technologies utilisées qui sont basées sur la programmation orientée objet, et puisque il existe plusieurs outils de conception qui le supporte. Nous allons utiliser pour la modélisation de notre projet un sous-ensemble nécessaire et suffisant du langage UML. Cet ensemble contient les diagrammes les plus utilisé et les plus connus tel que le diagramme de cas d'utilisation qui représente les fonctionnalités du système du point de vue de l'utilisateur, le Diagramme de séquence c'est un diagramme d'interactions UML, le diagramme de classes qui est le point central dans un développement orienté objet et le diagramme de déploiement qui représente l'utilisation de l'infrastructure physique.[12]

4.2 Processus de développement logiciel

Le choix d'un processus de se fera en prenant en compte des critères essentiels de la plateforme à concevoir. Pour des raisons d'efficacement, de rapidité et d'analyse complète, nous opterons en effet pour n processus situé à mi-chemin entre UP (Unified Process) et XP (eXtrema Programming) c'est l'AUP (Agile Unifid Process) il est conduit par les cas d'utilisation, comme UP, mais beaucoup plus simple. Il est relativement léger et restreint, comme XP, mais sans négliger les activités de modélisation en analyse et conception. Il est fondé sur l'utilisation d'un sous-ensemble nécessaire et suffisant du langage UML.

4.3 Présentation d'AUP (Agile Unified Process)

AUP est une version simplifiée de l'IBM Rational Unifies Process (RUP), développé par Scott Ambler. Il décrit une démarche, simple et facile de comprendre, de développement des logiciels commerciaux en utilisant des techniques agiles et concepts fidèles à la RUP. L'AUP applique des techniques agiles y compris le développement piloté par les tests (TDD), la

modélisation agile, la gestion du changement agile, et ré-factorisation de bases de données pour améliorer la productivité.

5 Chronogramme de réalisation

Cette figure présente le déroulement chronologique des étapes de notre stage de fin d'études. Durant les deux premiers mois, nous avons effectué les taches préliminaires comme la documentation, l'étude technique, l'étude du contexte du projet, ensuite nous nous sommes concentrés à la méthodologie de travail et l'étude des besoins pour arriver à la spécification des besoins et l'élaboration des diagrammes de cas d'utilisation. Vers le troisième mois, nous avons élaboré la partie conception de notre application, à savoir les diagrammes de classe et de séquences. Le mois de mai a été consacré pour le développement.

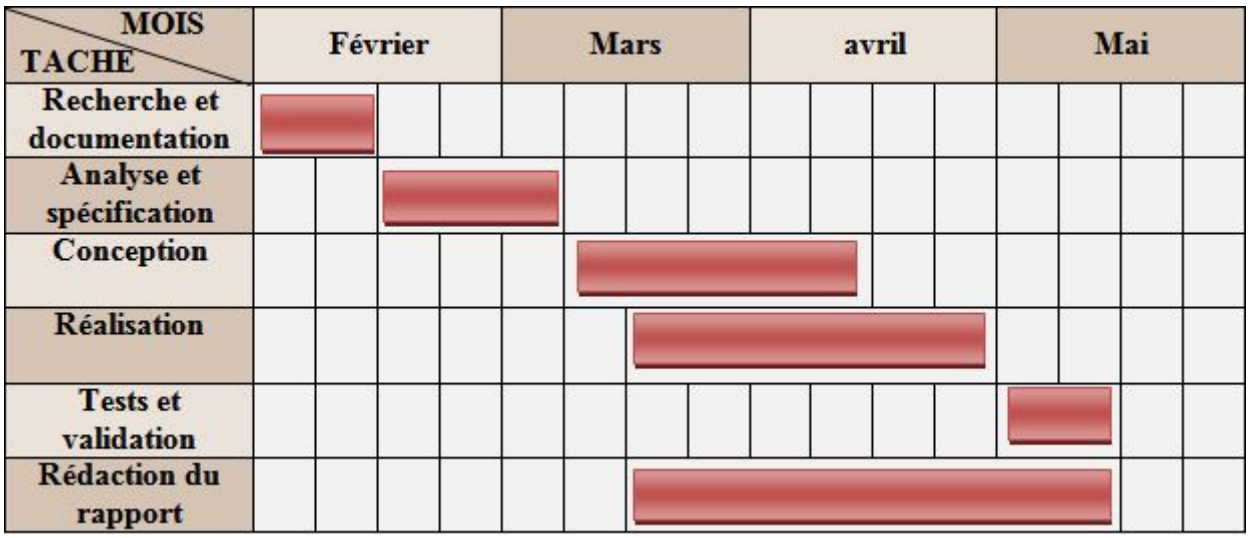

FIGURE I.3 – Chronogramme

Conclusion

Après avoir placé notre travail dans son contexte, précisé la méthodologie à suivre pour satisfaire nos besoins, nous allons définir, dans le chapitre suivant, les besoins fonctionnels et non fonctionnels qui vont servir à la mise du projet dans son contexte.

Chapitre II

DÉFINITION ET SPÉCIFICATION DES BESOINS

Introduction

L'ingénierie des besoins est l'une des activités les plus importantes puisqu'elle permet de déterminer les besoins de l'utilisateur de l'application et définir les contraintes envisagées. Dans ce chapitre, nous allons étudier en premier lieu les outils existants, nous allons ensuite présenter les besoins fonctionnels et non fonctionnels pour spécifier les cas d'utilisation. Enfin, nous élaborons une maquette d'interface homme-machine (IHM) qui fera l'objet d'un modèle de base des interfaces que nous allons implémenter par la suite.

1 Analyse des besoins

L'analyse fonctionnelle est une démarche qui consiste à rechercher et à caractériser les fonctions offertes par un produit pour satisfaire les besoins de son utilisateur.

1.1 Besoins fonctionnels

Les besoins fonctionnels sont les fonctionnalités et les actions que le système doit obligatoirement effectuer, ils sont issus du cahier de charges du projet après un consensus avec le client (entreprise d'accueil).

L'application à réaliser doit permettre aux éventuels utilisateurs :

- La génération de la base de données (création et mise à jours des tables, création et mise à jour des déclencheurs).
- La génération des fichiers de la classe Business (les méthodes CRUD, surcharge de fonctions).
- la génération des interfaces (les vues).
- la génération des classes métier (les contrôleurs).
- l'utilisation des fichiers de configuration et structure XML.

Ces besoins seront illustres à travers l'identification des cas d'utilisations.

1.2 Besoins non fonctionnels

Un besoin non fonctionnel est un besoin spécifiant des priorités du système, telles les contraintes liées à l'environnement, à l'implémentation et les exigences en matière de performance, de dépendance de plateformes, de facilité de maintenance, d'extensibilité et de fiabilité.

Ainsi cet application doit répondre à un ensemble de fonctionnalités qui ne sont pas explicitement demandées par l'utilisateur mais qui contribuent à l'amélioration de l'application, à savoir :

- La création de la structure et la configuration XML à partir d'une base de données existante.
- Le déploiement facile : l'application (ORM) doit être facile à déployer, elle ne doit pas nécessiter l'installation d'autres outils et elle n'impose à son utilisateur aucun changement. De plus, l'application doit être facilement paramétrable.
- L'ergonomie : L'application doit être facile à accéder et ne demande pas un temps d'apprentissage. L'interface doit être conviviale afin d'inciter les utilisateurs à adopter cet outil.
- La sécurité et la gestion d'accès : un aspect crucial qui doit être présent dans l'application pour ne pas permettre l'accès à n'importe quel utilisateur. Pour cela, une étape d'authentification doit être implémentée afin de sécuriser cet accès.
- L'élaboration d'un manuel d'utilisation décrivant la façon d'utiliser l'application.
- L'extensibilité : Possibilité d'ajout ou de modification de nouvelles fonctionnalités.

2 Solution proposée

Les acteurs et les cas d'utilisation sont les concepts UML fondamentaux pour la spécification des exigences. Dans cette partie, nous allons les identifier à partir de l'expression des besoins, nous allons ensuite les représenter graphiquement à l'aide du diagramme de cas d'utilisation principal et ses raffinements.

2.1 Identification des acteurs

Un acteur représente l'abstraction d'un rôle joué par des entités externes (utilisateur, dispositif matériel ou autre système) qui interagissent directement avec le système.

Nous avons détecté l'existence de quatre acteurs qui interviennent dans les processus de notre application qui sont :

- L'utilisateur simple de l'application c'est le client de l'entreprise.
- Le développeur web qui réalise des applications destinées à ces utilisateurs simples, pour ceci, il utilise des langages de programmation tel que PHP, HTML et d'autres, afin de réaliser sa tache principale qui est la création des fichiers de configuration pour chaque nouvelle application.
- Le système de vérification est le système responsable de la vérification de la cohérence de notre structure des fichiers XML et de la base de données.
- Le système d'exécution est l'outil qui permet de générer l'environnement applicatif final de l'application.

2.2 Identification des cas d'utilisations

Les attentes des utilisateurs ont été spécifiées et pour mieux comprendre les besoins fonctionnels nous allons présenter les fonctionnalités principales du système sous forme de cas d'utilisation qui regroupent une famille de scénarios possibles selon un critère fonctionnel. Un scénario correspond au flot de messages échangés entre l'acteur et le système. Le résultat sera un diagramme de cas d'utilisation global et une description textuelle de chaque cas d'utilisation.

2.2.1 Diagramme de cas d'utilisation général

Le diagramme représenté par la figure ci-dessous présente une vue globale sur le projet et les différentes fonctionnalités offertes.

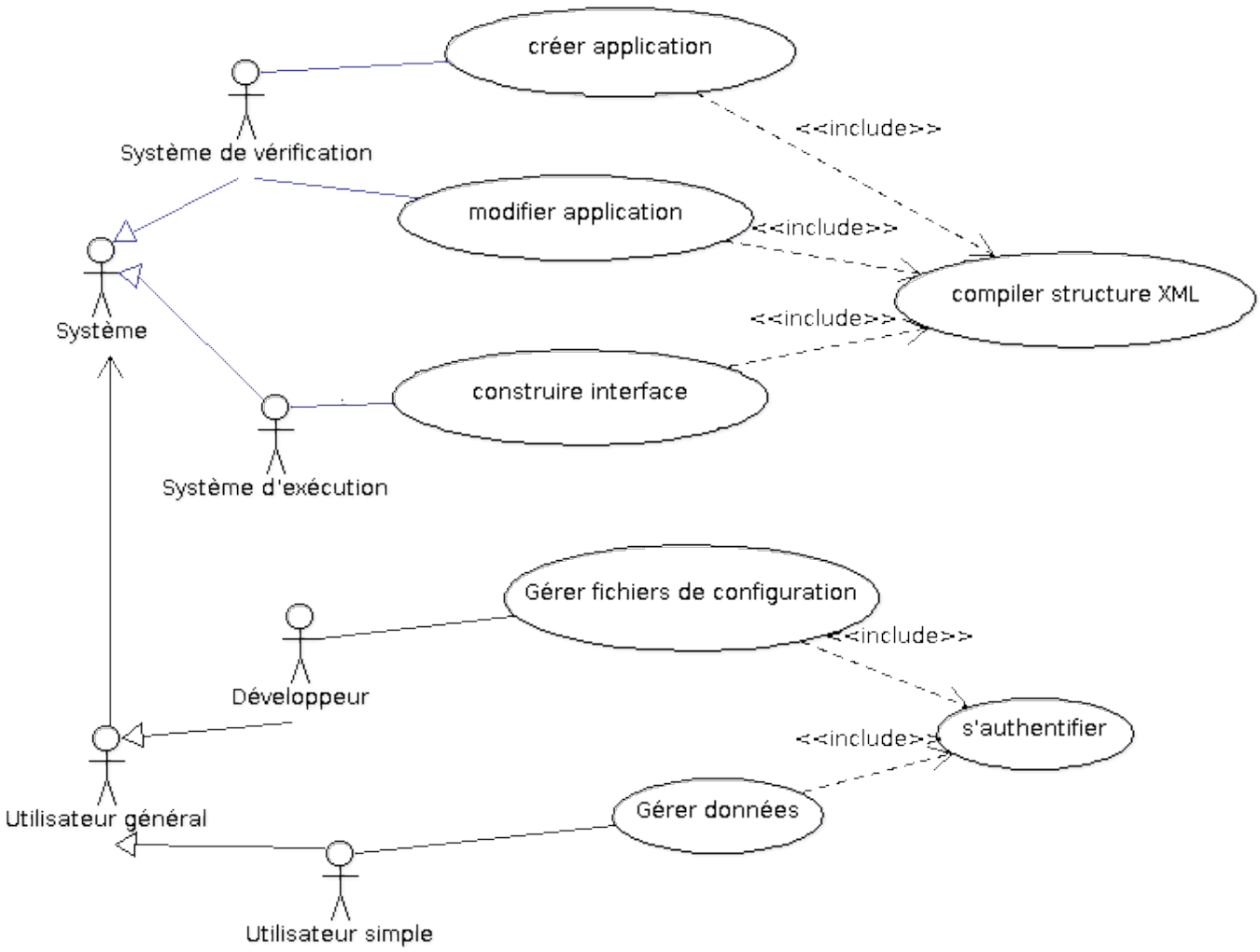

FIGURE II.1 – Diagramme de cas d'utilisation général

Notre système interroge les différentes couches de l'application pour garantir la communication entre les différentes couches du système.

Nous avons traduit à travers le diagramme de cas d'utilisation général divers taches réalisées par les acteurs notamment le système qui peut être un système de vérification ou bien un système d'exécution, et l'utilisateur général qui est un développeur ou bien un utilisateur simple :

- Le cas d'utilisation « s'authentifier » permet l'authentification des utilisateurs d'ORM afin d'accéder à la plateforme.
- Le cas d'utilisation « créer application » concerne le système de vérification ce dernier va créer une nouvelle application après la compilation des fichiers de configuration.
- Le cas d'utilisation « modifier application » concerne le système de vérification qui va

faire les modifications nécessaires sur la structure des classes et la structure de la base de données.

- Le cas d'utilisation « construire interface » concerne le système d'exécution, celui la vas créer des nouvelles interfaces à partir des applications ou projets existants.
- Le cas d'utilisation « compiler structure XML » est une couche utilisée par tous les acteurs système de notre applications ces derniers compilent les fichiers de configuration XML utilisant l'analyseur lexical, syntaxique et sémantique.
- Le cas d'utilisation « Gérer fichiers de configuration » concerne les utilisateurs développeurs de notre ORM, qui peuvent donner aux utilisateurs simples le droit d'ajouter,modifier ou supprimer des fichiers de configuration.
- Le cas d'utilisation « Gérer données » concerne l'utilisateur simple de l'application générée par notre ORM, cette tache est traduite par la modification, la suppression ou l'ajout les données de l'application.

2.2.2 Cas d'utilisation « Compiler structure XML »

Le diagramme de cas d'utilisation suivant représente les différentes actions possibles pour compiler les fichiers des configurations XML.

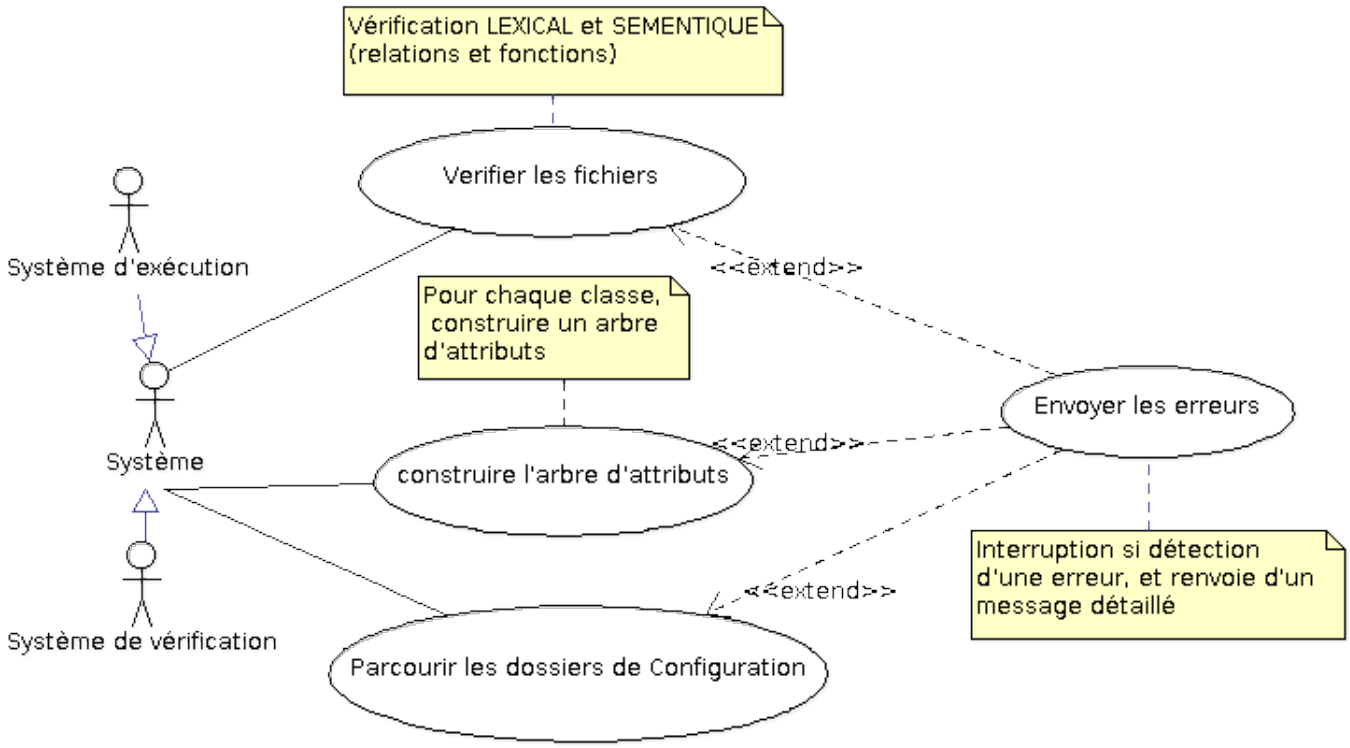

FIGURE II.2 – Diagramme de cas d'utilisation « Compiler structure XML »

Cas d'utilisation :	compiler structure XML
Acteur :	Système (système de vérification et système d'exécution)
Pré condition :	Existantes des fichiers de configuration XML
Post condition :	Compiler les données et les rendre lisibles par le système
Description du scenario principale :	- Parcourir les dossiers de configuration - Analyser les fichiers XML (semantic, lexical et syntaxique) - Construire un arbre d'attributs

TABLE II.1 – Description du cas d'utilisation « Compiler structure XML »

2.2.3 Cas d'utilisation « Créer application »

Le diagramme de cas d'utilisation ci-dessous représente les différents taches permettant la création d'une nouvelle application, pour mener bien ces tache le système doit créer une nouvelle base de données ainsi que les contraintes et les déclencheurs correspondants aux fichiers de configuration XML. Par la suite, il procède à l'insertion des nouveaux utilisateurs à la table "USER".

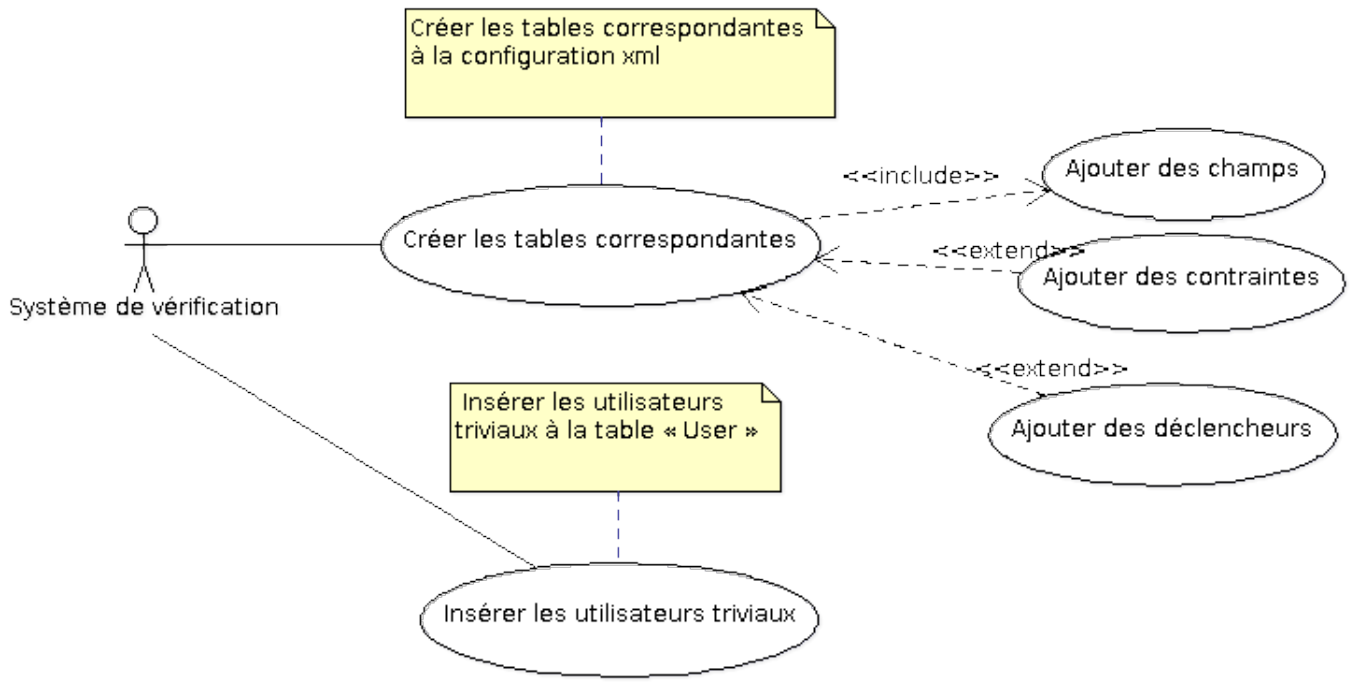

FIGURE II.3 – Diagramme de cas d'utilisation « Créer application »

Cas d'utilisation :	créer application
Acteur :	Système de vérification
Pré condition :	Existences des fichiers de configuration XML
Post condition :	Créer les nouveaux fichiers des méthodes, base de données, insérer les utilisateurs triviaux.
Description du scenario principale :	Après la compilation et les vérifications des erreurs le système de vérification vas créer une nouvelle application correspondante aux fichiers de configuration.

TABLE II.2 – Description du cas d'utilisation « Créer application »

2.2.4 Cas d'utilisation « Modifier application »

Le diagramme de Cas d'utilisation ci-dessous représente les différentes actions possibles pour modifier une application déjà existante dans notre système.

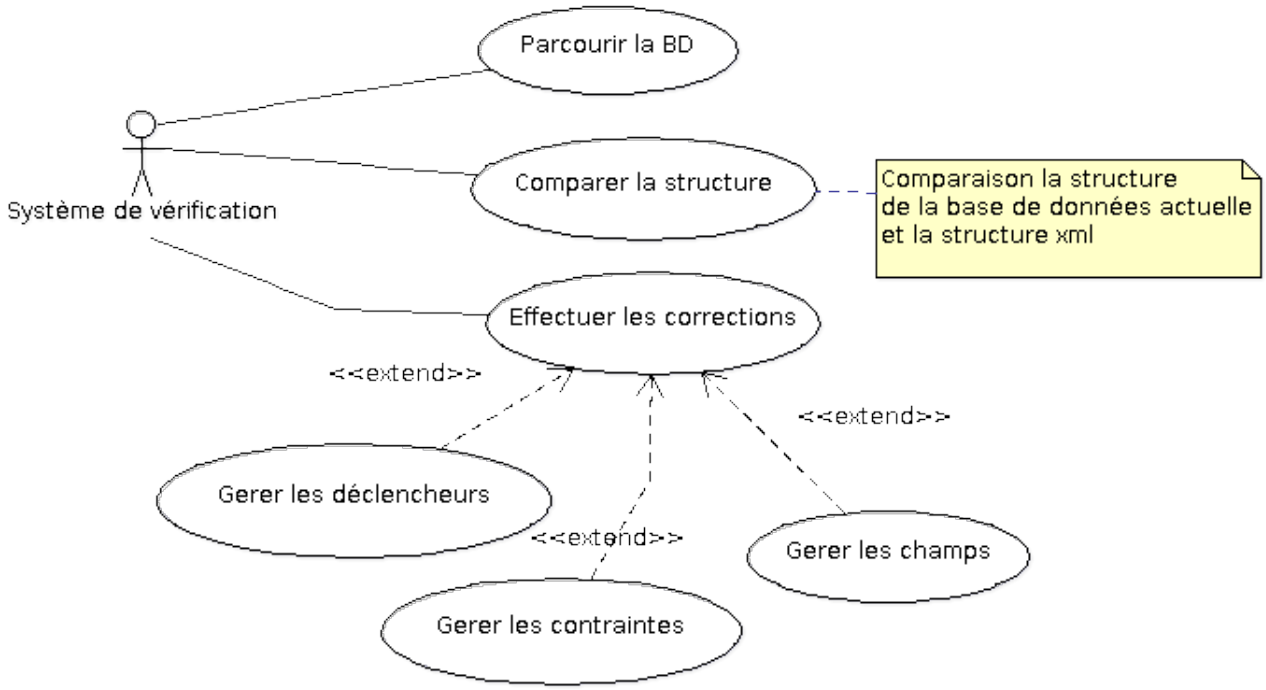

FIGURE II.4 – Diagramme de cas d'utilisation « Modifier application »

Cas d'utilisation :	Modifier application
Acteur :	Système de vérification
Pré condition :	Existences des modifications sur les fichiers de configuration XML (date de modification du fichier)
Post condition :	Modifier une application existante.
Description du scenario principale :	Dans ce cas le système de vérification doit : – Comparer la structures XML et la structure déjà installée (BD) – Effectuer les corrections nécessaires sur les déclencheurs, les contraintes et les champs de la base de données

TABLE II.3 – Description du cas d'utilisation « Modifier application »

2.2.5 Cas d'utilisation « Construire interface »

Le diagramme de cas d'utilisation ci-dessous représente les différentes taches du système d'exécution pour générer les interfaces de l'application que le système de vérification à demandé de le traiter.

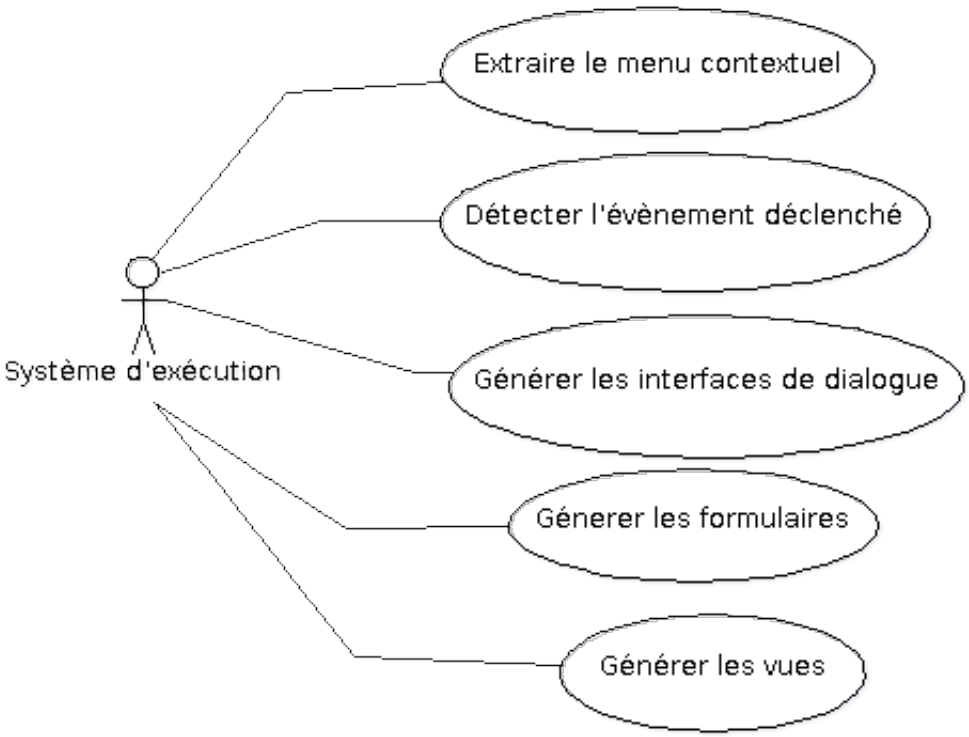

FIGURE II.5 – Diagramme de cas d'utilisation « Construire interface »

Cas d'utilisation :	Modifier application
Acteur :	Système d'exécution
Pré condition :	L'existante d'une application et leur fichiers de configuration XML
Post condition :	Générer tous les interfaces Homme machine (IHM).
Description du scenario principale :	Dans cet cas le système d'exécution doit : – Extraire le menu contextuel – Détecter l'événement déclencher par l'utilisateur – Générer les interfaces de dialogues, les formulaires et les vues

TABLE II.4 – Description du cas d'utilisation « Construire interface »

2.2.6 Cas d'utilisation « Gérer fichiers de configuration »

Le développeur a le droit de gérer les fichiers de configuration d'une application bien déterminé. Cette tache consiste à l'ajout, suppression et la modification et nécessite une authentification.

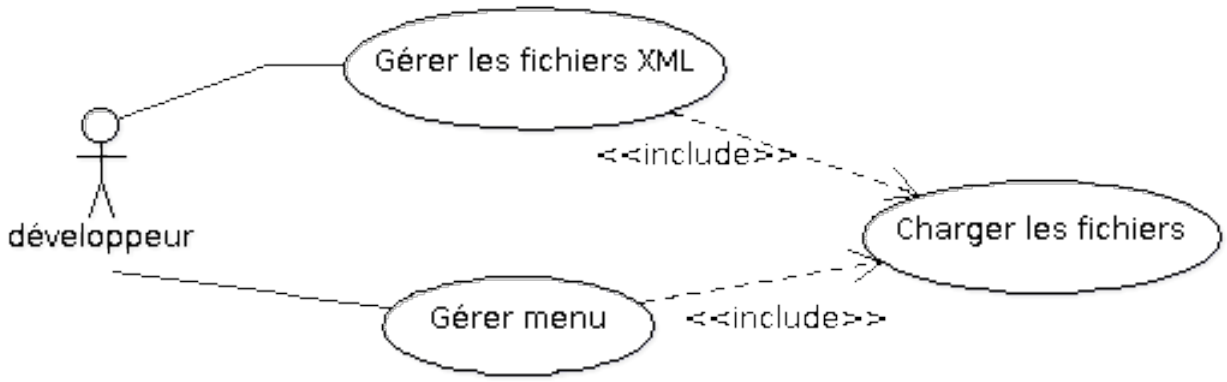

FIGURE II.6 – Diagramme de cas d'utilisation « Gérer fichiers de configuration »

Cas d'utilisation :	Gérer fichiers de configuration
Acteur :	Utilisateur principale d'application
Pré condition :	L'existante d'un compte FTP pour télécharger les fichiers de configuration dans le serveur WEB.
Post condition :	Cette tâche permet de gérer : – Le noyau de l'application – Le menu principal destiné aux utilisateurs simples de l'application (le surcharge des menus par défaut)
Description du scenario principale :	Dans ce cas le développeur doit : – Gérer les fichiers de configuration (ajouter, modifier, supprimer) – Gérer le menu principal (ajouter,supprimer,modifier les champs ou les liens, ...)

TABLE II.5 – Description du cas d'utilisation « Gérer fichiers de configuration »

2.2.7 Cas d'utilisation « Gérer données »

Les utilisateur simples des applications contrôlées par le système ORM peuvent gérer leurs propres données en s'authentifiant avec un nom d'utilisateur et mots de passe pour accéder aux différentes interfaces de la plateforme.

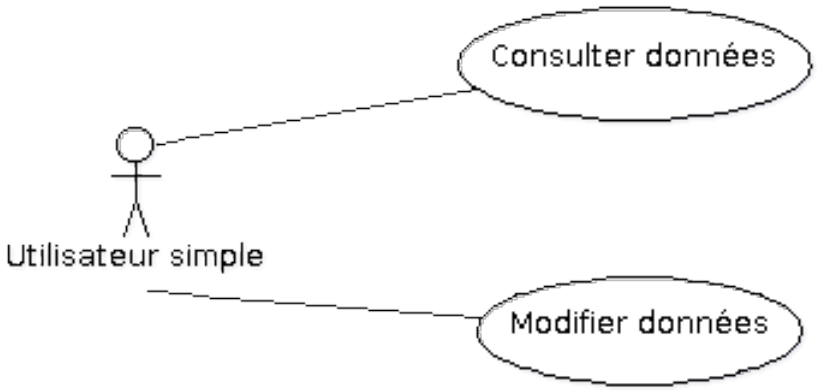

FIGURE II.7 – Diagramme de cas d'utilisation « Gérer données »

Cas d'utilisation :	Gérer données
Acteur :	Utilisateur principale d'application
Pré condition :	L'existante d'une application dans le système, et d'un nom d'utilisateur et d'un mot de passe.
Post condition :	Effectuer les modifications nécessaire sur les données en cas de besoin.
Description du scenario principale :	La tâche de gestion de données permet de : – Consulter les interfaces. – Consulter les formulaires de l'application. – Modifier ou supprimer les données.

TABLE II.6 – Description du cas d'utilisation « Gérer données »

2.2.8 Cas d'utilisation « S'authentifier »

Les deux acteurs (utilisateur simple et développeur) de notre système et leurs applications, doivent de s'authentifier pour accéder aux interfaces de la plateforme a fin de charger les fichiers de configuration dans le serveur web. Dans la figure ci dessous illustre les tâches possibles permettant l'authentification d'un utilisateur.

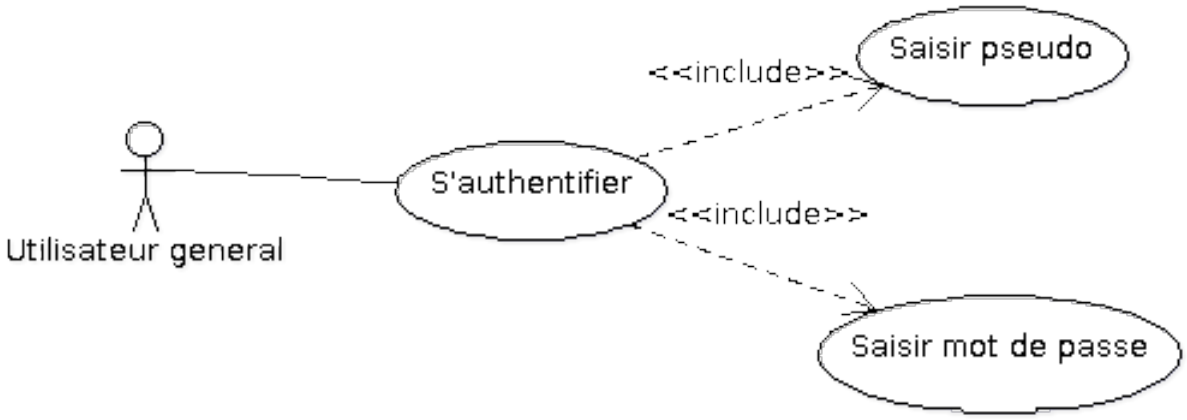

FIGURE II.8 – Diagramme de cas d'utilisation « S'authentifier »

Cas d'utilisation :	S'authentifier
Acteur :	utilisateur simple et utilisateur développeur
Pré condition :	L'utilisateur saisit ses droits d'accès (pseudo et mot de passe).
Post condition :	Sécuriser l'accès
Description du scenario principale :	– L'utilisateur saisit ses droits d'accès. – Le système vérifie si les champs sont vides ou non. – Le système affiche un message en cas d'erreur d'authentification. – Le système redirige l'acteur vers son espace selon son rôle.

TABLE II.7 – Description du cas d'utilisation « S'authentifier »

3 Maquette IHM

Après avoir modélisé les besoins fonctionnels, nous allons représenté une maquette IHM (Interface Homme-Machine) qui schématise une vue globale mais non définitive de la structure de la future interface de qui va-être générée et contrôler automatiquement par notre ORM.

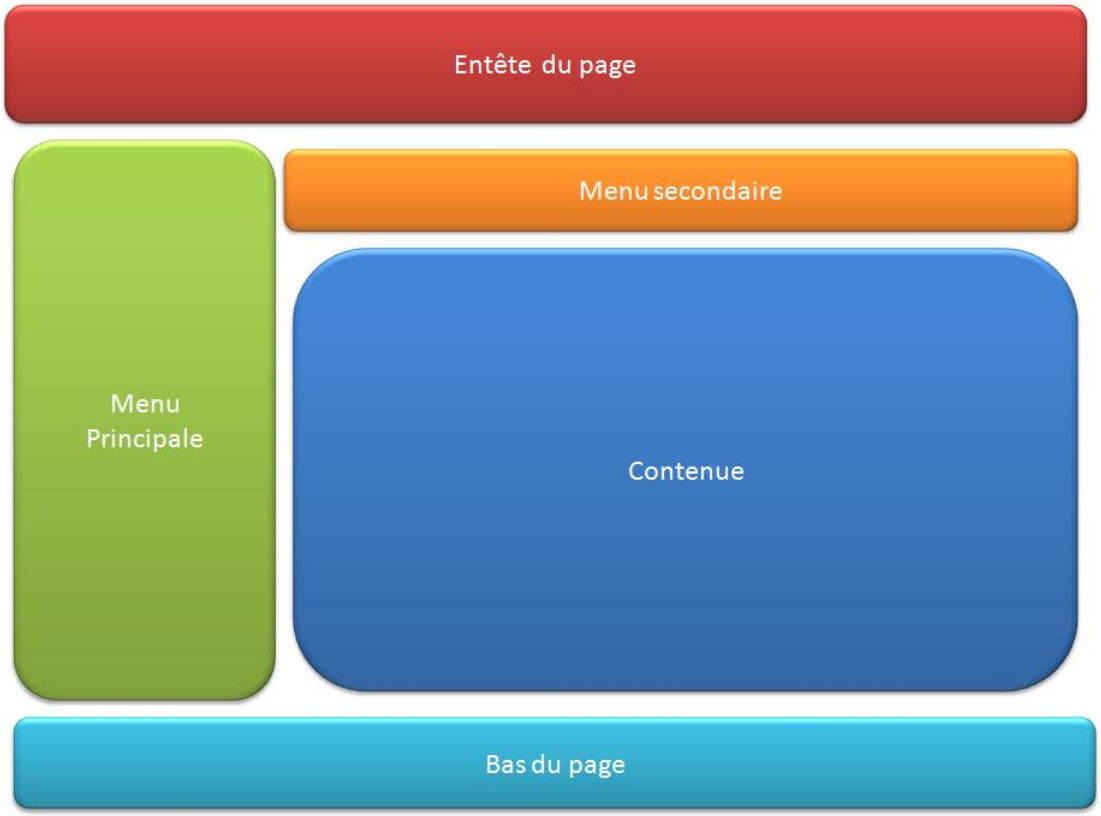

FIGURE II.9 – Maquette IHM par défaut

Conclusion

Dans ce chapitre nous avons détaillé les besoins fonctionnels et non fonctionnels de notre projet. Cette étude facilitera la tâche de la conception que nous allons détailler dans le chapitre suivant.

Chapitre III

CONCEPTION

Introduction

Dans ce chapitre, nous allons présenter l'étude conceptuelle de notre projet que nous allons illustrer par une vue dynamique contenant les diagrammes d'activités et de séquences puis une vue statique du système consolidée par le diagramme de classe et le diagramme de déploiement.

1 Vues Dynamiques

Le but de la conception dynamique est de donner des scénarios de déroulement des différentes tâches de l'application. Nous avons opté, pour la conception dynamique de notre projet, aux diagrammes de séquences et d'activités.

1.1 Diagrammes de séquences

Nous avons choisi les séquences les plus importantes afin de simplifier la compréhension de l'ensemble de l'application.

Le Diagramme de séquence est un diagramme d'interactions UML. Ils représentent des échanges de messages entre les éléments, dans le cadre d'un fonctionnement particulier du système. Nous l'utiliserons pour monter l'ordre chronologique des messages envoyés et reçus par un ensemble d'objets.

1.1.1 S'authentifier

L'authentification des utilisateurs d'application ou d'un système ORM est obligatoire pour vérifier leurs identités afin de leurs autoriser l'accès aux ressources du système, la figure ci-dessous montre l'enchaînement du processus d'authentification :

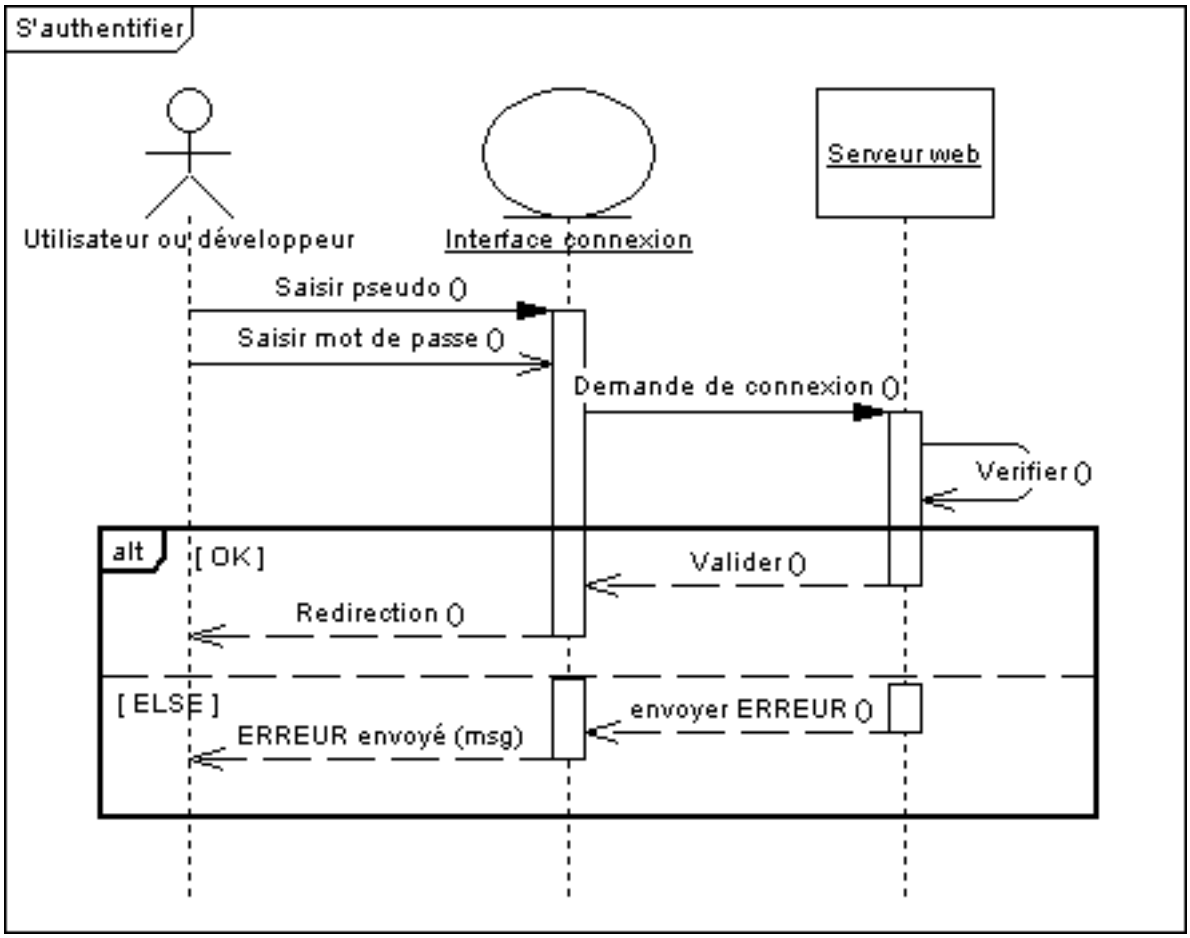

FIGURE III.1 – Diagramme de séquence « S'authentifier »

Chacun des acteurs doit saisir son nom d'utilisateur et son mot de passe. Le système vérifie les informations saisies au niveau du serveur web, si elles sont correctes, le système le redirige vers son espace selon son rôle, sinon un message d'erreur sera affiché.

Dans le cas d'un acteur développeur, le mode d'authentification proposé est plus sécurisé, basé sur le protocole FTP, sur lequel il envoie les fichiers, d'où une authentification est nécessaire à chaque connexion aux serveurs.

1.1.2 Compiler structure XML

La couche compilation est importante dans notre projet pour transformer le code source de balisage écrit en XML à un langage PHP5 et aux Requêtes SQL pour créer une nouvelle application.

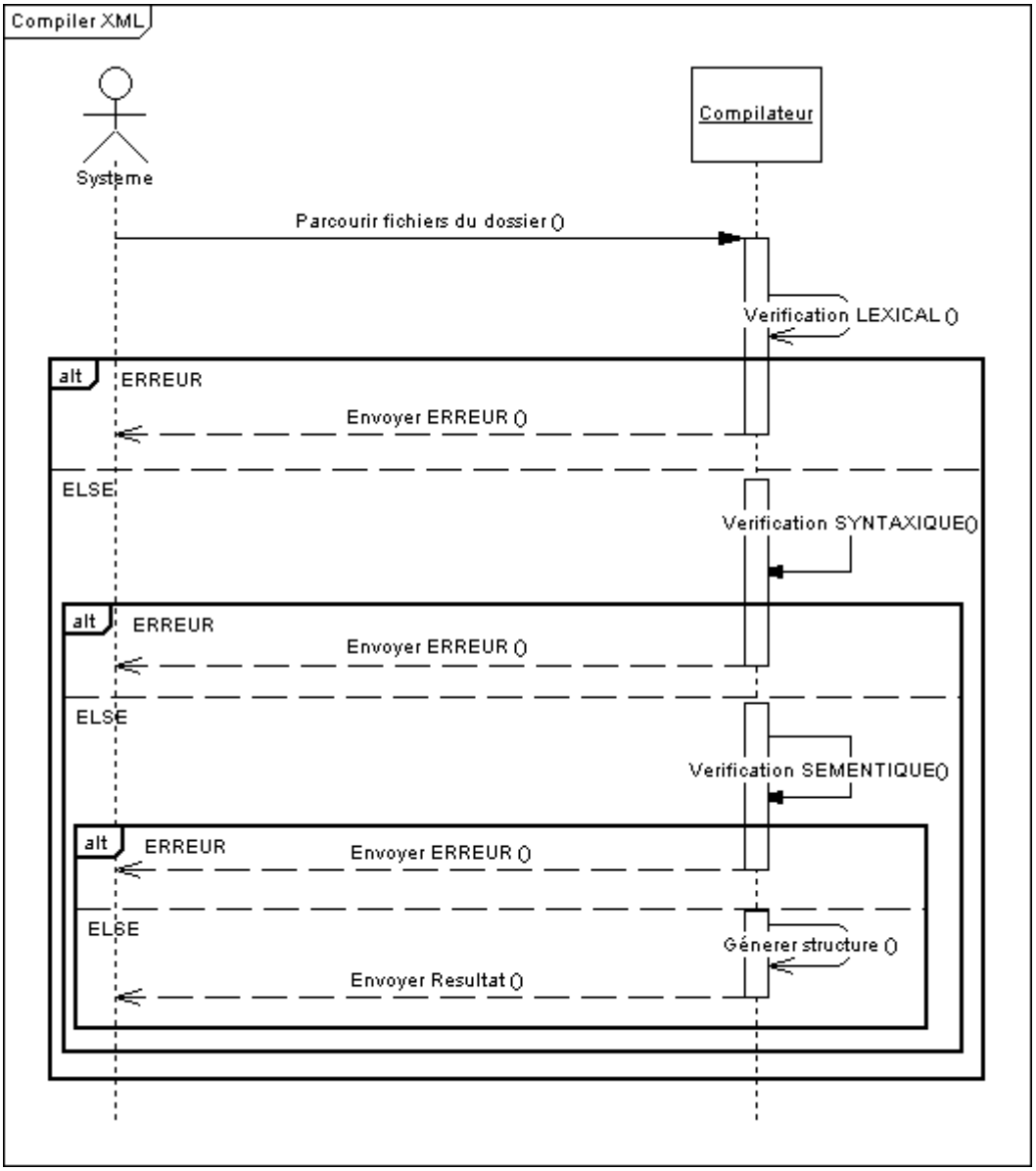

FIGURE III.2 – Diagramme de séquence « Compiler structure XML »

Pour compiler les fichiers XML et les rendre lisibles par le système, l'utilisateur déclenche un événement de parcours au compilateur qui interprète les données, vérifie la partie lexicale et sémantique et s'assure que chaque fonction appelée dans le programme est compréhensible, sinon il génère un message d'erreur. Pour chaque classe, cette phase de compilation prend en charge la construction des arbres d'attributs. En cas d'interruption, elle renvoie un message d'erreur. Finalement le compilateur constitue la phase finale pour obtenir une structure de l'application.

1.1.3 Créer Application

Le diagramme de séquence ci-dessous présente le cas d'utilisation « créer application »

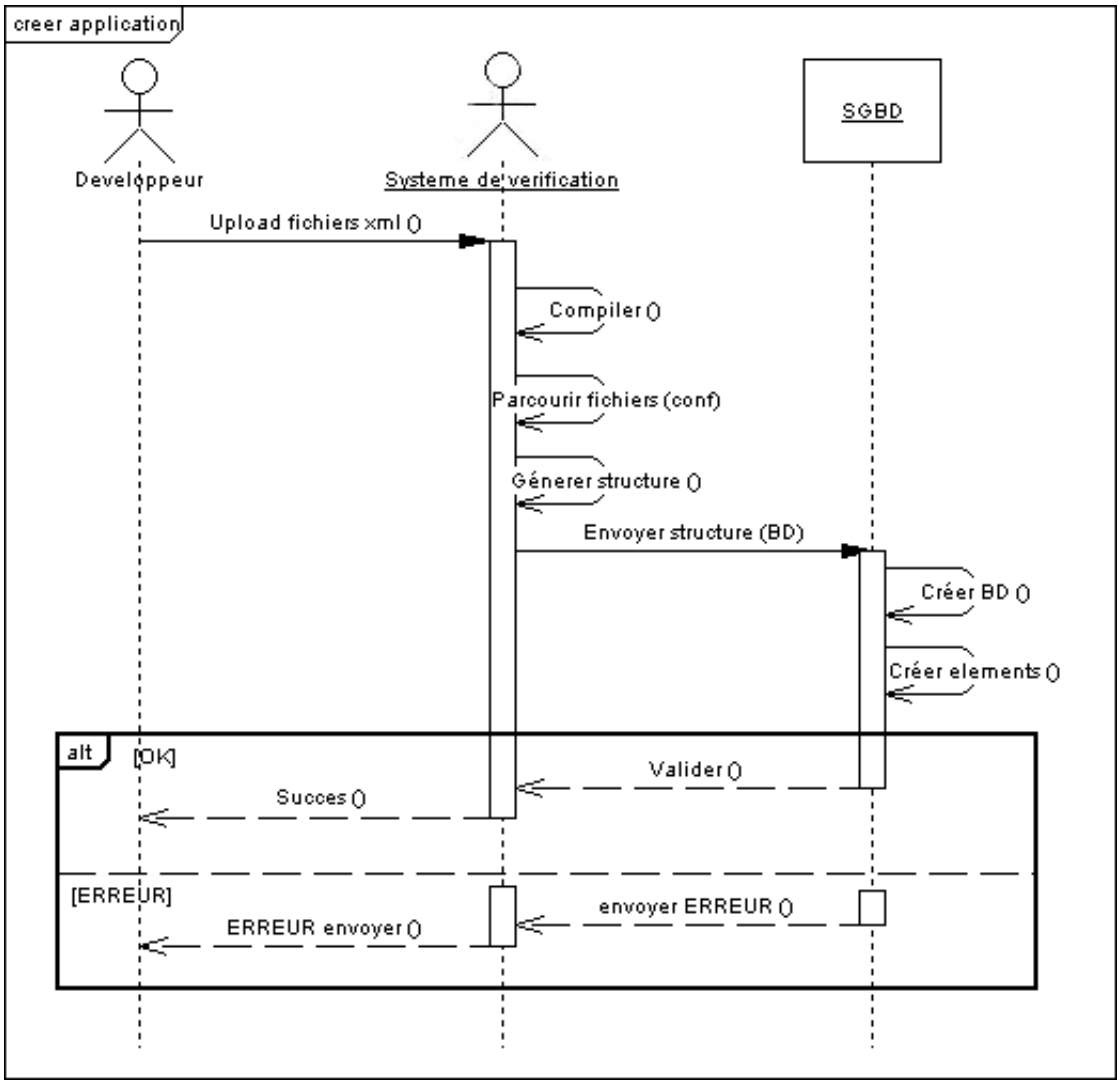

FIGURE III.3 – Diagramme de séquence « Créer Application »

Le diagramme de séquences présent fait entrer en action les instances de classes intervenants dans la création de l'application, en effet, le développeur charge les fichiers XML et déclenche l'événement faisant appel au système de vérification avec l'ensemble des (objet, classe, compilateur..) chacun correspondant à une sous fonction du système ayant comme rôle la compilation et le parcours des fichiers ainsi que la génération de la structure correspondante qu'il envoie au serveur de la base de données pour créer ces éléments. Ensuite un message d'information sera affiché s'il n'y a pas un bug détecté, sinon l'application créée sera validée avec succès.

1.1.4 Modifier application

La modification des applications est contrôlée par notre système. Le scénario décrivant le déroulement de cette tache est décrit par le diagramme de séquences suivant :

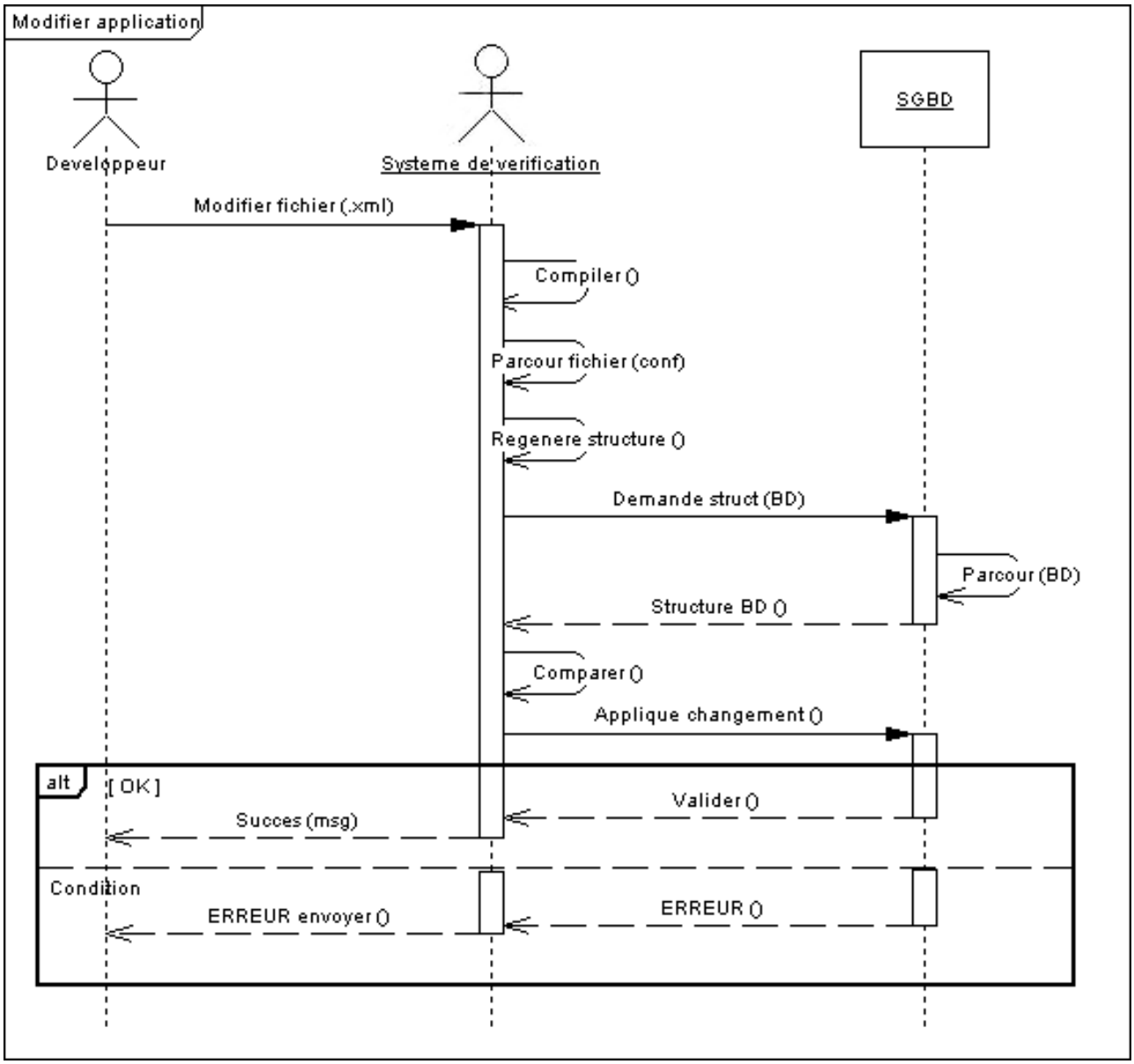

FIGURE III.4 – Diagramme de séquence « Modifier application »

Pour que l'utilisateur arrive a réaliser des modifications sur son application, les fichiers de configuration doivent passés par l'intermédiaire d'un système de vérification qui permet d'invoquer les fichier déjà téléchargés. En faisant appel aux fonctions et aux variables globales de librairie, l'entité vérification commence à parcourir les données en les compilant ,ensuite elle détecte les événements en vérifiant la structure de chaque objet. Finalement le système de vérification applique les changements nécessaires dans la base de données a travers une comparaison avec la nouvelle structure.

1.1.5 Construire interface

Dans notre système, nous allons créer des objets prédéfinis tel que Data-Grid, checkbox afin d'assurer la compilation rapide. Le diagramme de séquence ci-dessous décrit la construction des interfaces de notre application.

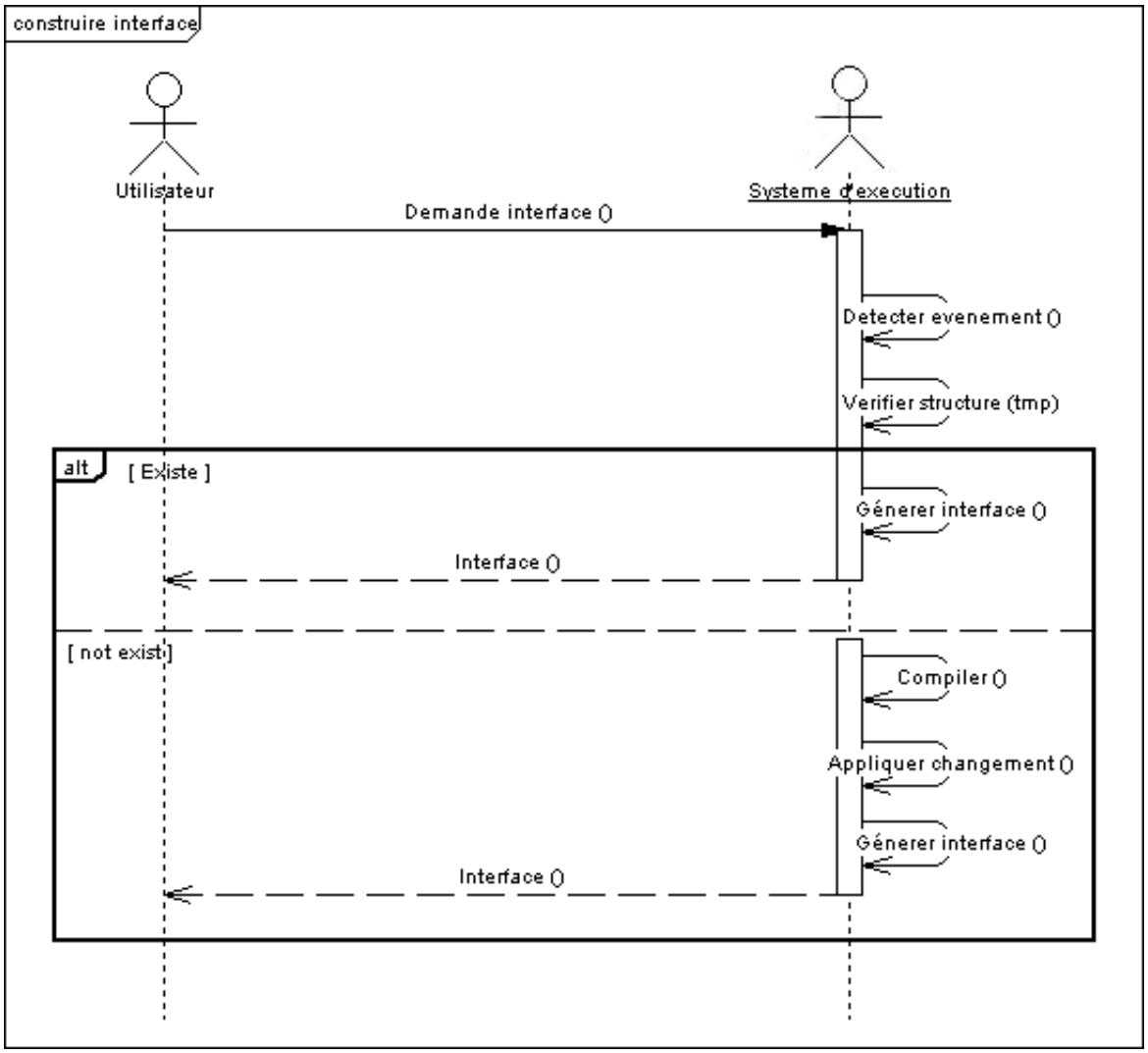

FIGURE III.5 – Diagramme de séquence « Construire interface »

Cette phase de construction d'interface concerne le système d'exécution, qui prend en charge la création des nouvelles interfaces à partir des applications ou projets existants. En premier lieu, ce système commence par extraire le menu contextuel et détecte l'événement déclenché en vérifiant la structure.

Après la vérification de l'existence d'une application et de leur fichiers de configuration XML, le système génère les interfaces de dialogue, les formulaires et les vues, dans le cas contraire il refait les mêmes étapes en effectuent les changements nécessaire.

1.1.6 Gérer fichiers de configuration

Le développeur peut gérer les fichiers de configuration et le menu de l'application, cette tache est mieux décrite à travers le scénario suivant :

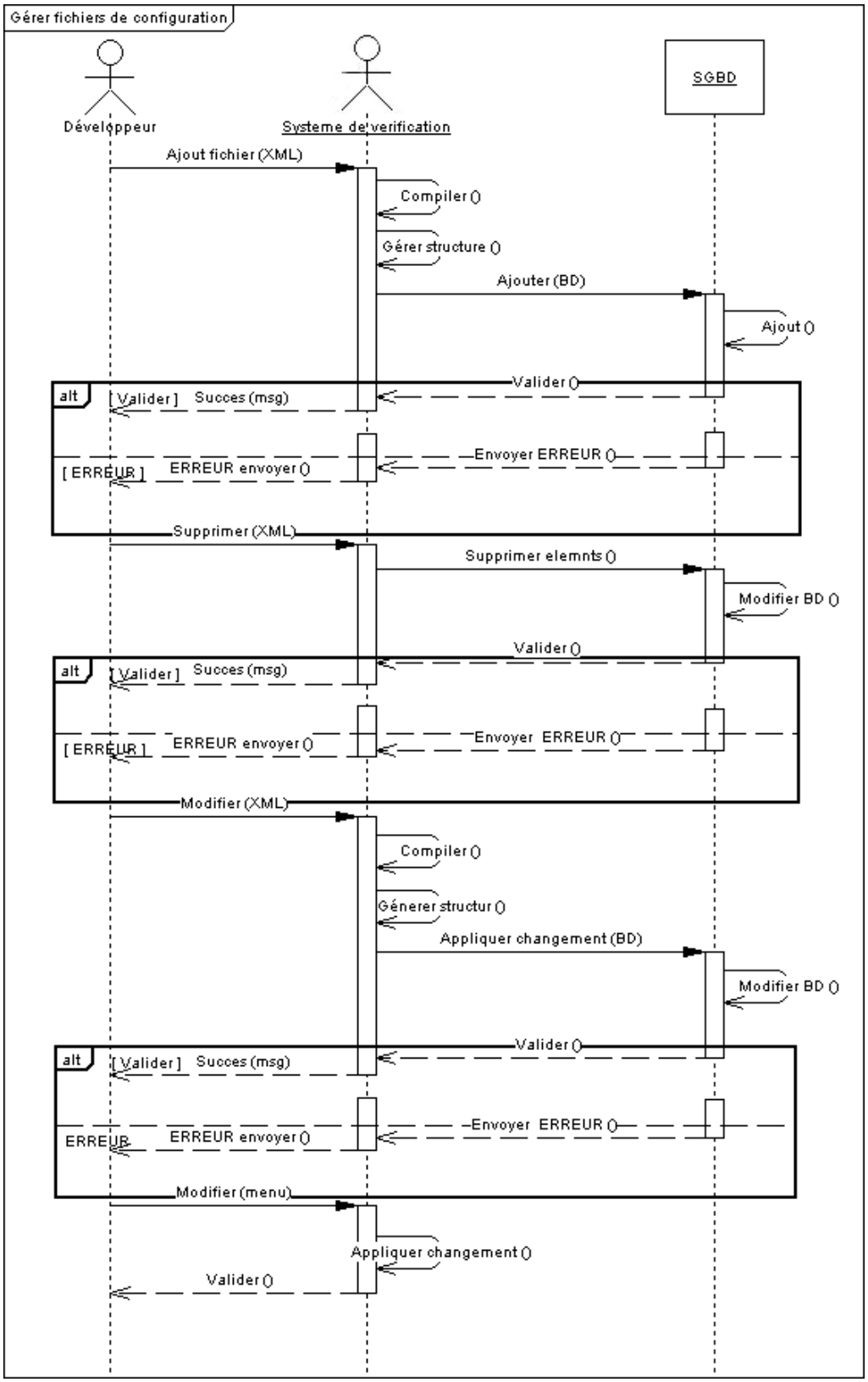

FIGURE III.6 – Diagramme de séquence « Gérer fichiers de configuration »

Suite aux chargement des fichiers de configurations XML dans le serveur WEB, l'utilisateur peut ajouter, supprimer et modifier ces fichiers avec le menu principal gère et compile les structures déjà générés et les ajoutes dans la base de données après avoir effectué les modifications imposés par l'acteur principal.

1.2 Diagrammes d'activités

Pour décrire le flux de travail (workflow), nous avons choisi le diagramme d'activités.(Figure III.7,Figure III.8)

1.2.1 Diagramme d'activités général

Les interventions des différents acteurs sur la plateforme sont détaillées à travers le diagramme d'activités général. Le système interprète les actions d'utilisateur principal de l'application afin d'assurer la cohérence et l'intégrité des données.

1.2.2 Diagramme d'activités « Compiler structure XML »

En effet, à chaque opération de compilation, le système parcourt les fichiers de configuration XML, dont la structure est maintenue par le DTD correspondant, et génère l'arbre d'attributs permettant de mettre à jour la structure de la base de données et les interfaces HTML.

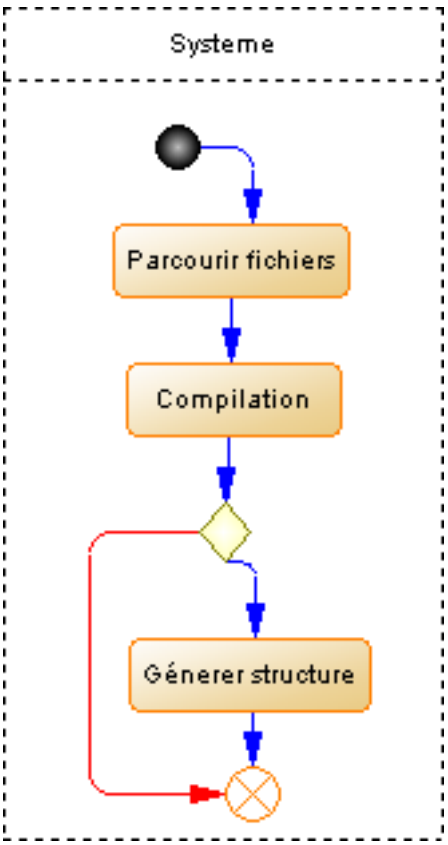

FIGURE III.8 – Diagramme d'activités « Compiler structure XML »

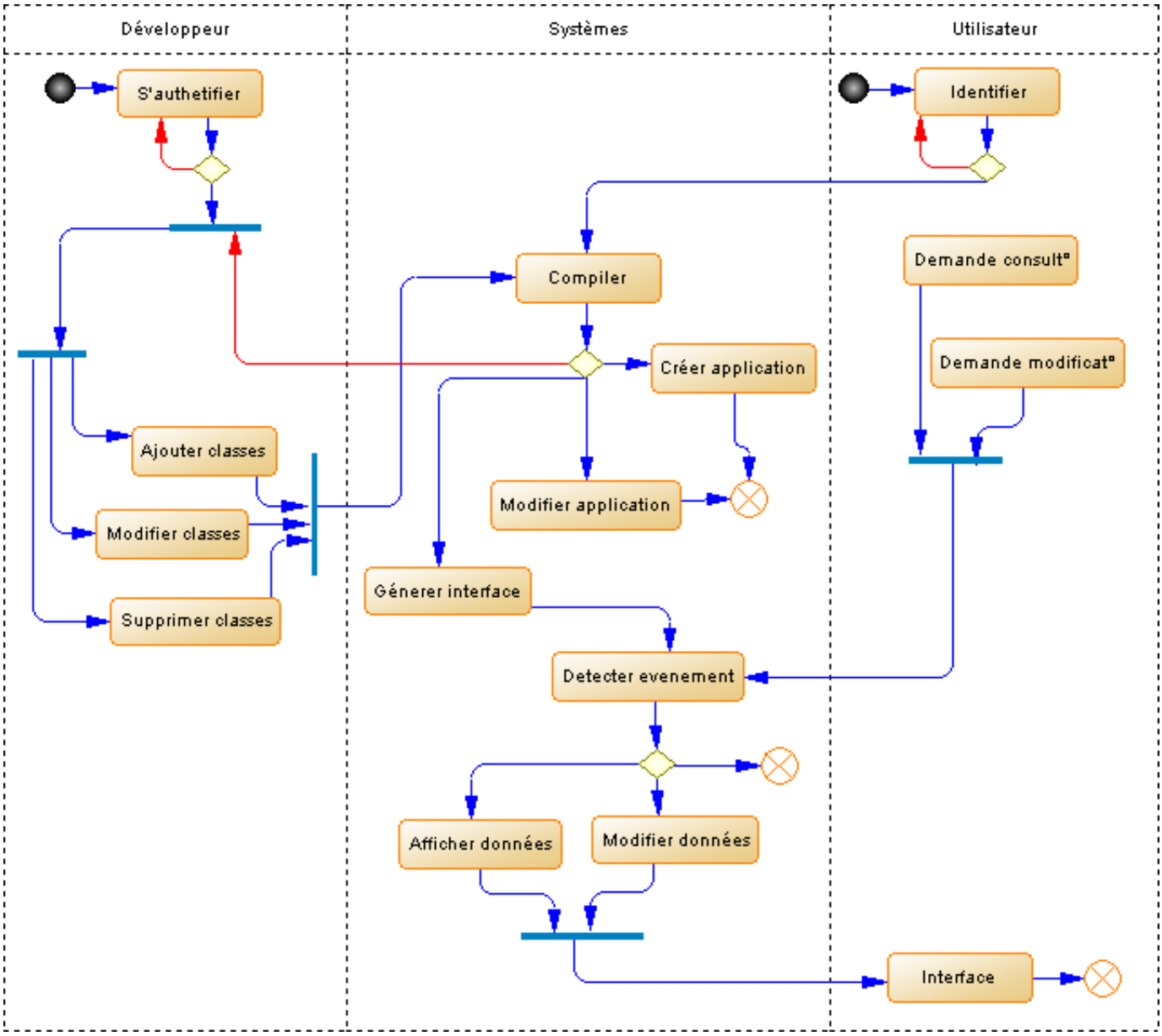

FIGURE III.7 – Diagramme d'activités général

2 Vues Statique

Le terme vue statique s'applique à ces aspects d'un modèle donné ayant un rapport avec les types de choses qui existent dans le modèle, leur structure interne et les relations qui existent.

Pour décrire la vue statique de notre application, nous avons choisi le diagramme de classes ainsi que le diagramme de déploiement.

2.1 Diagramme de classes

Le diagramme de classes présente le point central dans un développement orienté objet. En analyse, il a pour objectif de décrire la structure des entités manipulées par les utilisateurs.

Nous avons ainsi ajouté les méthodes nécessaires à chaque classe. En pratique, celles-ci servent d'interfaces entre l'ORM (côté serveur) et la couche présentation (côté client).

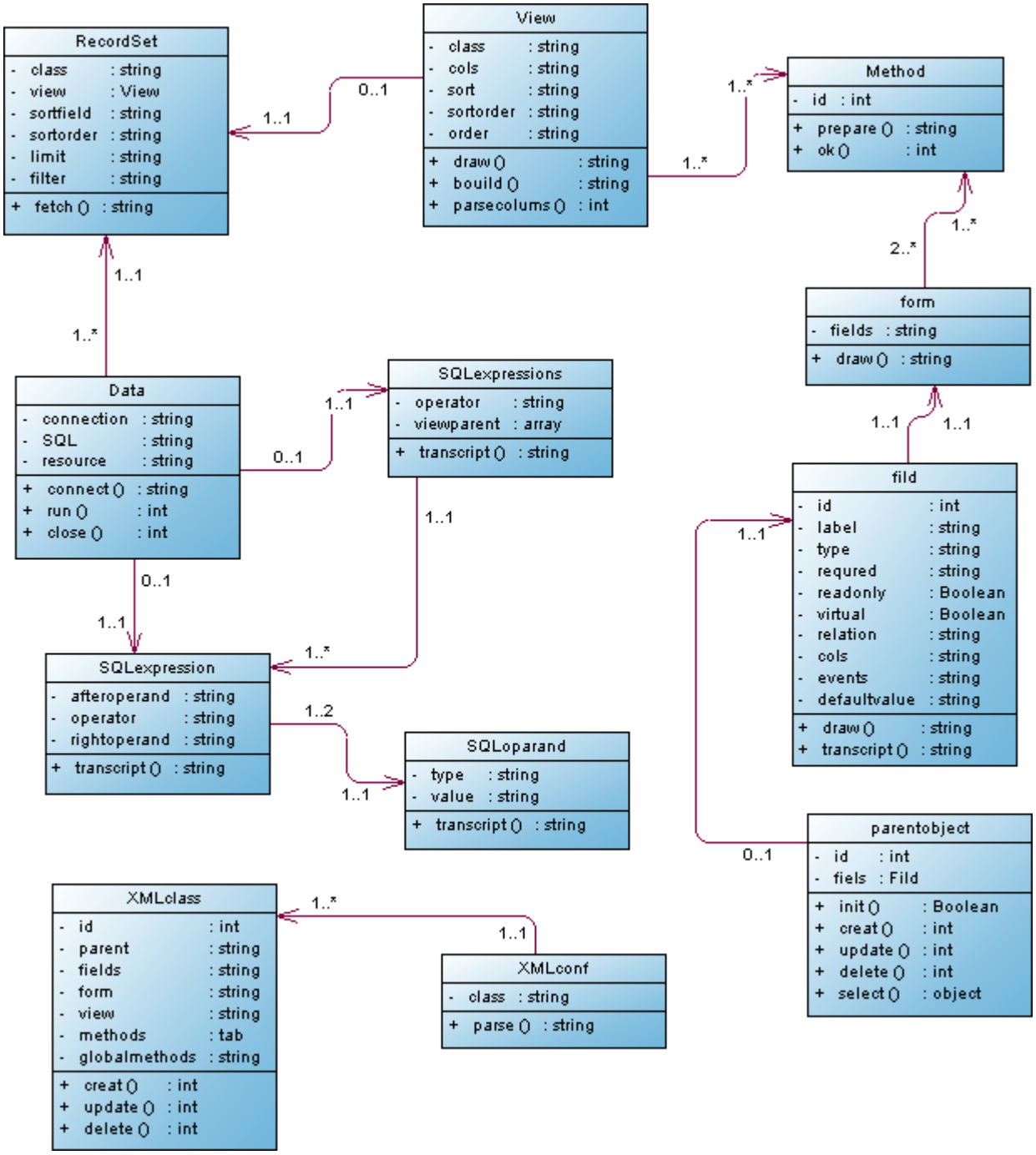

FIGURE III.9 – Diagramme de classes

Ces classes concernent le cœur de la plateforme, ils permettent de gérer le modèle de

l'application finale, à travers la classe ParentObject.

Cette classe assure la liaison entre le noyau de l'ORM et toute autre application fille. Elle contient tous les attributs relatifs aux objets à créer comme l'identifiant unique, ainsi que les méthodes CRUD. Le lien se fait à travers des relations d'héritage.

Cette même classe fait appel à une XMLClass qui définit l'objet généré à partir de la structure XML, à savoir les vues, les formulaires et les champs etc.

Les vues et les formulaires et les méthodes sont définis à travers les classes View, Form et Method, ils permettent respectivement de formater une liste de données, afficher un enregistrement et effectuer une action sur un ou plusieurs objets.

La classe Data gère la connexion à la base de données et l'exécution des requêtes en contrôlant les erreurs générées par le système de gestion de base de données. Elle permet de créer des RescordSet, en utilisant des SQLExpressions.

2.2 Diagramme de déploiement

Le diagramme de déploiement sert à représenter l'utilisation de l'infrastructure physique par le système et la manière avec laquelle les composants du système sont répartis ainsi que leurs relation entre eux. Les éléments utilisés par un diagramme de déploiement sont principalement les nœuds, les composants, les associations et les artefacts. Les caractéristiques des ressources matérielles et des supports de communication peuvent être précisées par stéréotype.

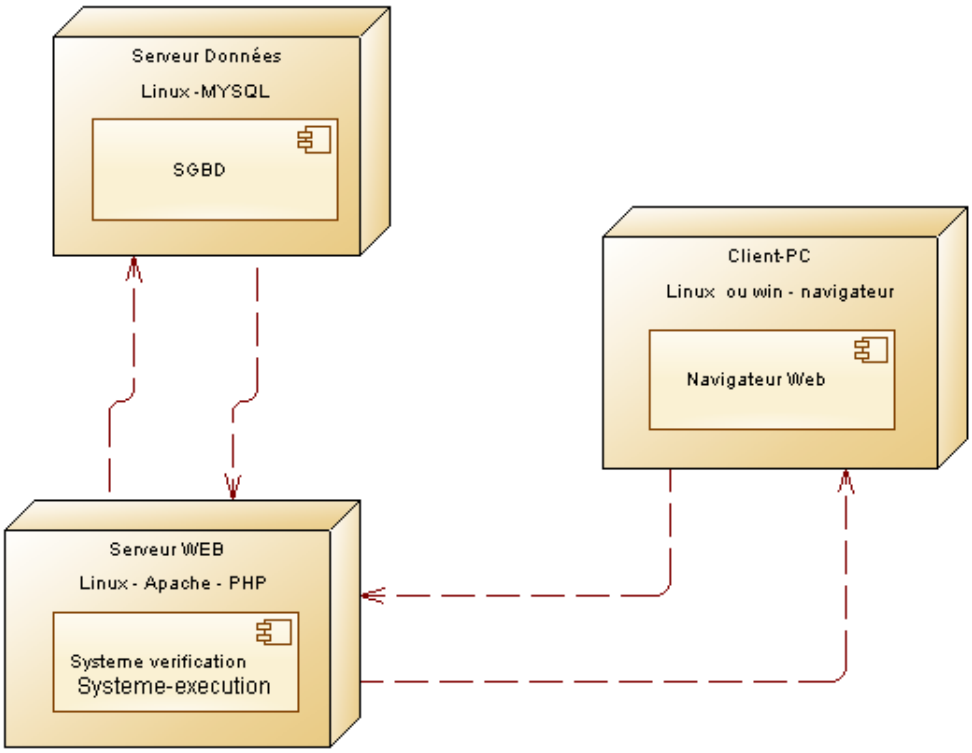

FIGURE III.10 – Diagramme déploiement

La modélisation de ce diagramme de déploiement ci dessus montre trois nœuds :

- Serveur web : Composé d'un module de scripts PHP.
- Serveur données :Contient le système de gestion de base de données MySQL.
- Client-PC : Représente le navigateur utilisé pour visualiser les interfaces.

Conclusion

Au cours de ce chapitre nous avons pu dégager une vue conceptuelle qui englobé toutes les fonctionnalités majeures à réaliser. La nécessité d'une telle étude se révèle très utiles lors de la phase d'implémentation.

Chapitre IV

RÉALISATION

Introduction

Ce chapitre traite des aspects techniques liés à l'implémentation et la mise en œuvre de notre système S.R.M (système objet relationnel).
Nous commençons donc par la spécification des différents outils technologies adoptés pour le développement de notre système. Nous aborderons ensuite les principales fonctionnalités offertes par notre application et nous finissons par dernier phase d'implémentation à travers l'illustration des étapes de la réalisation qui vont résumer le fonctionnement global de notre système (S.R.M).

1 Choix technologique

Après la définition de la liste des principales fonctionnalités que nous voulons mettre en place, les caractéristique que notre plateforme doit posséder et le coût de mise en place, nous avons choisi les technologies qui nous aide à développer notre application (S.R.M).

1.1 PHP 5

PHP (Hypertext Preprocessor) est un langage de programmation interprété libre principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif disposant depuis la version 5 de fonctionnalités de modèle objet complètes.[10]
Nous avons choisir PHP car il est simple à mettre en place (niveau serveur) et plus facile à standardiser, à transporter d'une application à une autre.

1.2 HTML 5

Le HTML5 se focalise sur les applications Web et l'interactivité, sans toutefois délaisser l'accessibilité et la sémantique. Le HTML5 se positionne également comme concurrent des technologies Flash et Silverlight.
L'avantage que nous ont pousser à choisi d'utiliser cette technologie c'est l'optimisation sur le temps de chargement des pages. HTML5 offre encore des nouvelles balises "sémantiques" permettant notamment de sectoriser plus facilement une page (en-tête <header>, menu <nav>, ou encore pied de page <footer>) et l'interactivité, <canvas> permet de créer des

animations. Enfin le HTML5 augmente l'expérience utilisateur avec la machine en permettant à des applications de s'exécuter en mode hors-ligne (offline) (stockage de données puis synchronisation lorsque la connexion est établie).[3]

1.3 JSON

JSON (Javascript Object Notation) est un format léger d'échange de données, il est indépendant de tout langage et il est reconnu nativement par Javascript. C'est une alternative à XML citons[4] quelques avantages pour les quelles nous avons choisie JSON :

- La vitesse de traitement
- La simplicité de mise en œuvre

1.4 JQUERY

jQuery est une bibliothèque qui permet d'agir sur le code HTML, CSS, JavaScript et AJAX,il permet de manipuler les éléments mis en place en HTML (textes, images, liens, vidéos, etc.) et mis en forme en CSS (position, taille, couleur, transparence, etc.) en utilisant des instructions simples qui donnent accès aux immenses possibilités de JavaScript et d'AJAX.[5]

1.5 CSS3

CSS (Cascading Style Sheets : feuilles de style en cascade) est un langage informatique qui sert à décrire la présentation des documents HTML et XML. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). Introduit au milieu des années 1990, CSS devient couramment utilisé dans la conception de sites web et bien pris en charge par les navigateurs web dans les années 2000.[**?**]

2 Choix architectural

Dans cette partie nous allons définir les modèles architectural que nous avons choisi afin d'assurer le fonctionnement de la partie logiciels de notre système et la partie matérielles.

2.1 Le modèle MVC

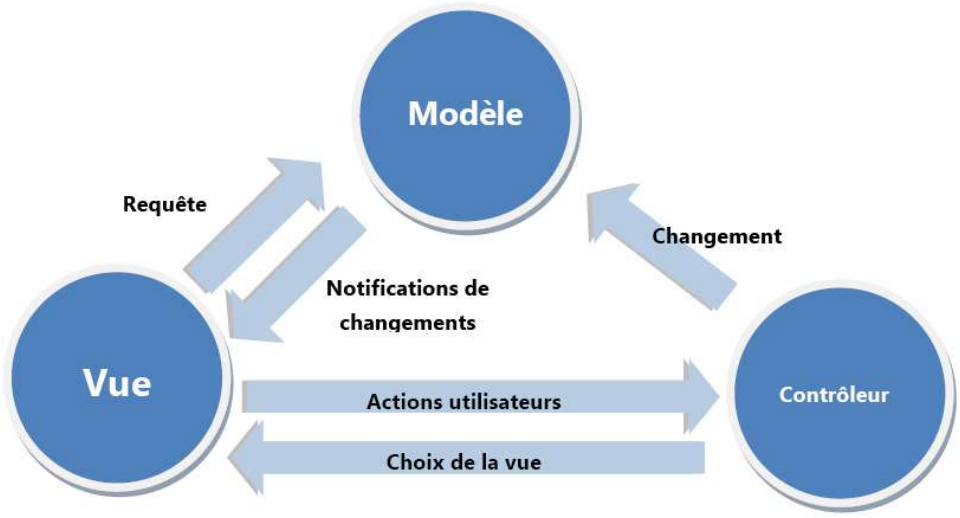

FIGURE IV.1 – Le modèle MVC

Ce modèle d'architecture impose la séparation entre les données, la présentation et les traitements, ce qui donne trois parties fondamentales dans l'application finale : le modèle, la vue et le contrôleur :[6]

2.1.1 Le modèle

Le modèle représente le comportement de l'application : traitements des données, interactions avec la base de données, etc. Il décrit ou contient les données manipulées par l'application. Il assure la gestion de ces données et garantit leur intégrité. Dans le cas typique d'une base de données, c'est le modèle qui la contient. Le modèle offre des méthodes pour mettre à jour ces données (insertion, suppression, changement de valeur). Il offre aussi des méthodes pour récupérer ces données. Les résultats renvoyés par le modèle sont dénués de toute présentation.

2.1.2 La vue

La vue correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur. Ces différents événements sont envoyés au contrôleur. La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle et d'interagir avec l'utilisateur.

2.1.3 Le contrôleur

Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle, ce dernier avertit la vue que les données ont changée pour qu'elle se mette à jour.

2.2 Architecture 3-tiers

Le style d'architecture en tiers permet de spécifier l'organisation des composants d'exploitation mis en œuvre pour réaliser l'application. Chaque partie indique une responsabilité technique à laquelle souscrivent les différents composants d'exploitation. Ce style d'architecture facilite la ré-utilisabilité au sein d'un système puisque les composants métiers correspondent à des concepts communs dans le cas présent.

Les différents tiers de système sont les suivants :

- Le stockage des données utilise un système de gestion de base de données (MYSQL).
- Le deuxième tiers est le middleware qui représente le service métiers, pour ce la nous avons utilisés le serveur web HHTP (Apache) qui à pour rôle de rendre accessible les informations du serveur web et d'exécuter les raquettes PHP.
- Les interfaces du notre système c'est le troisième tiers qui permet de visualiser les résultats finale qui sont générées à partir du notre serveur web.

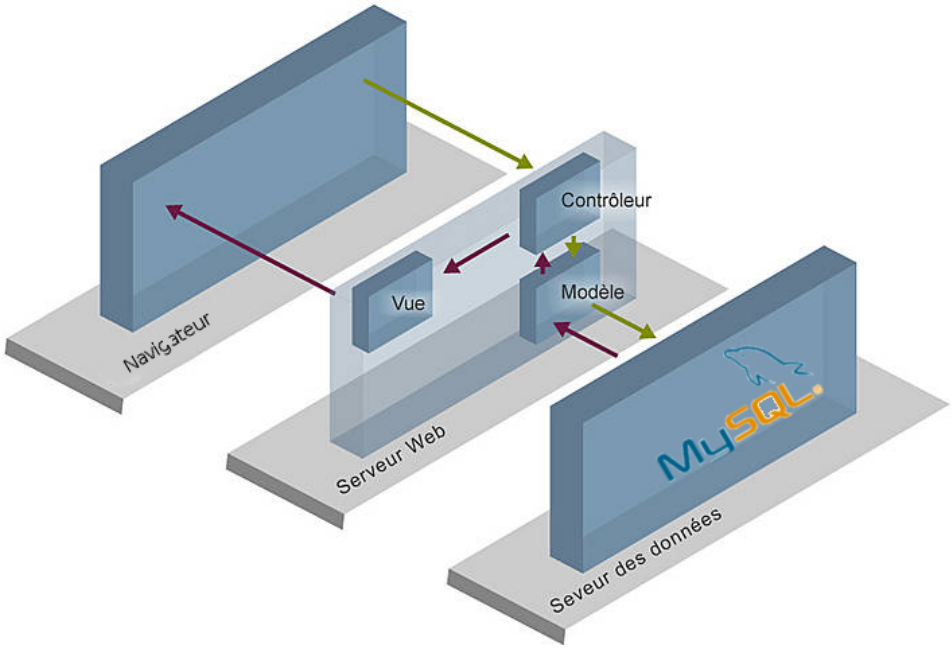

FIGURE IV.2 – Architecture 3-tiers

3 Environnement matériels

Dans cette partie nous allons identifier les outils matériels qui nous avons utilisées pour l'élaboration de notre projet.

3.1 Machines utilisés

Les machines utilisés pour que nous avons aboutir à ce projet sont :

- Un ordinateur portable dont les caractéristiques :
  - Processeur : Intel CORE 2 Duo
  - Mémoire RAM : 4 Go
  - Disque dur : 320 Go
  - Système d'exploitation : Fedora 18 (64-bits)
- Un ordinateur de bureau pour tester l'application dont les caractéristiques :
  - Processeur : Pentium 4
  - Mémoire RAM : 1 Go
  - Disque dur : 120 Go
  - Système d'exploitation : Windows XP (32-bits)

3.2 Architecture système

Cette architecture décrit l'envoi des raquettes au central (serveur), qui stocke toute les taches et les réclamations dans la base de données.

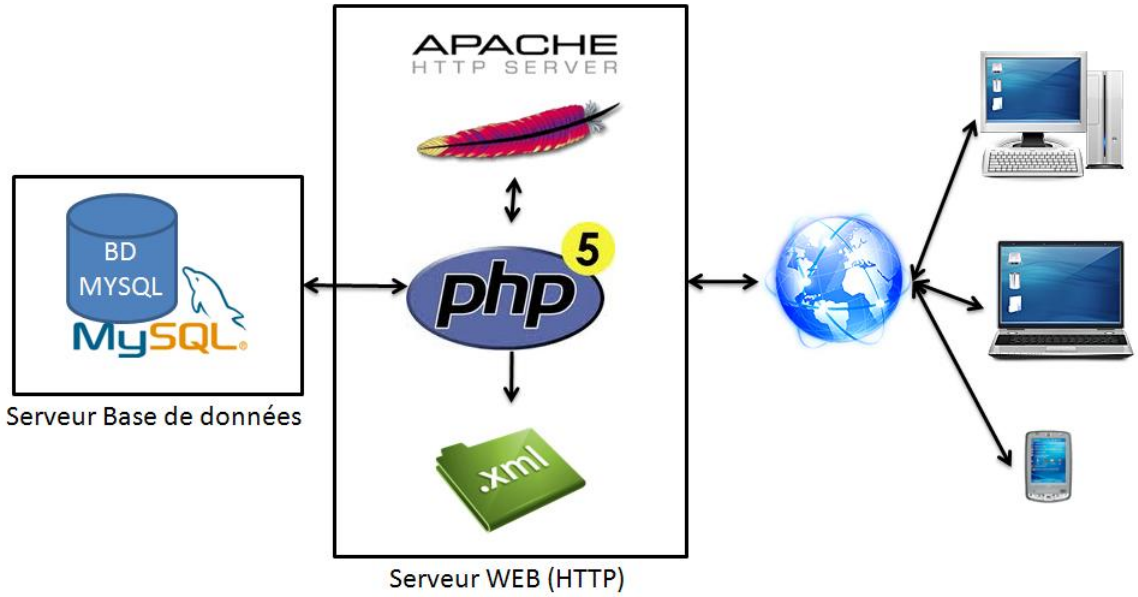

FIGURE IV.3 – Architecture logique du système

3.2.1 Serveur MYSQL

MySQL est un serveur de bases de données relationnelles SQL très rapide, multi-thread, robuste et multi-utilisateurs. MySQL est un logiciel libre développé sous double licence GPL (General Public License) et licence commerciale. Il est le serveur de base de données le plus utilisé dans le monde. Il fonctionne sur beaucoup de plates-formes différentes et il est accessible en utilisant plusieurs langages de programmation.[9]

3.2.2 Serveur HTTP Apache

Apache HTTP Server (Apache) est un serveur HTTP créé et maintenu au sein de la fondation Apache. C'est le serveur HTTP le plus populaire du World Wide Web.[1]

4 Environnement logiciel

Le choix de l'environnement logiciel vue qu'ils touche le facteur temps et le cout de l'application.

4.1 MySQL Workbench

MySQL Workbench est un outil visuel unifié destiné aux architectes de bases de données, aux développeurs et aux DBA. Il contribue à la modélisation des données et au développement de code SQL, et propose des outils d'administration complets pour la configuration des serveurs, la gestion des utilisateurs et bien plus encore,il est disponible sous Windows, Linux et Mac OS et il permet à un DBA, à un développeur ou à un architecte de données de concevoir, modéliser, générer et gérer visuellement des bases de données.[7]

Il comprend tout ce dont une personne impliquée dans la modélisation de données a besoin pour créer des modèles ER (entité-relation) complexes.

4.2 Filezilla

FileZilla Client (FileZilla) est un client FTP, FTPS et SFTP, à plusieurs fonctionnalités situant les principales :[8]

- Capacité à reprendre les mises à jour/téléchargements interrompus (si le serveur le supporte)

- Commandes personnalisables
- Réglage de la bande passante
- Système anti-déconnexion
- Détection des temps de pause pare-feu

4.3 L'environnement de développement NetBeans

Dans notre application on utilise la version 7, NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2 (Common Development and Distribution License). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring).

Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris, Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java).

Ce sont les avantages cités si-dessus et autres qui nous on amenés à choisir Netbeans comme environnement de développement pour notre système.

5 Scénarios d'utilisation

Dans cette partie, nous allons exposer quelques interfaces Homme/machine et machine/système pour montrer les services offerts par notre système. En faite S.R.M (système objet relationnel) offre des interfaces simples, lisibles et faciles à manipuler. Ceci est assuré par une structuration bien organisée des différentes fonctionnalités.

5.1 Authentification développeur

Le développeur doit s'authentifier au serveur web avec un compte FTP ou SFTP pour créer un nouveau dossier qui contient tous les fichiers nécessaires pour générer une nouvelle application. Par la suite il procède au chargement (UPLOAD) des fichiers de configuration XML dans le nouveau dossier créé.

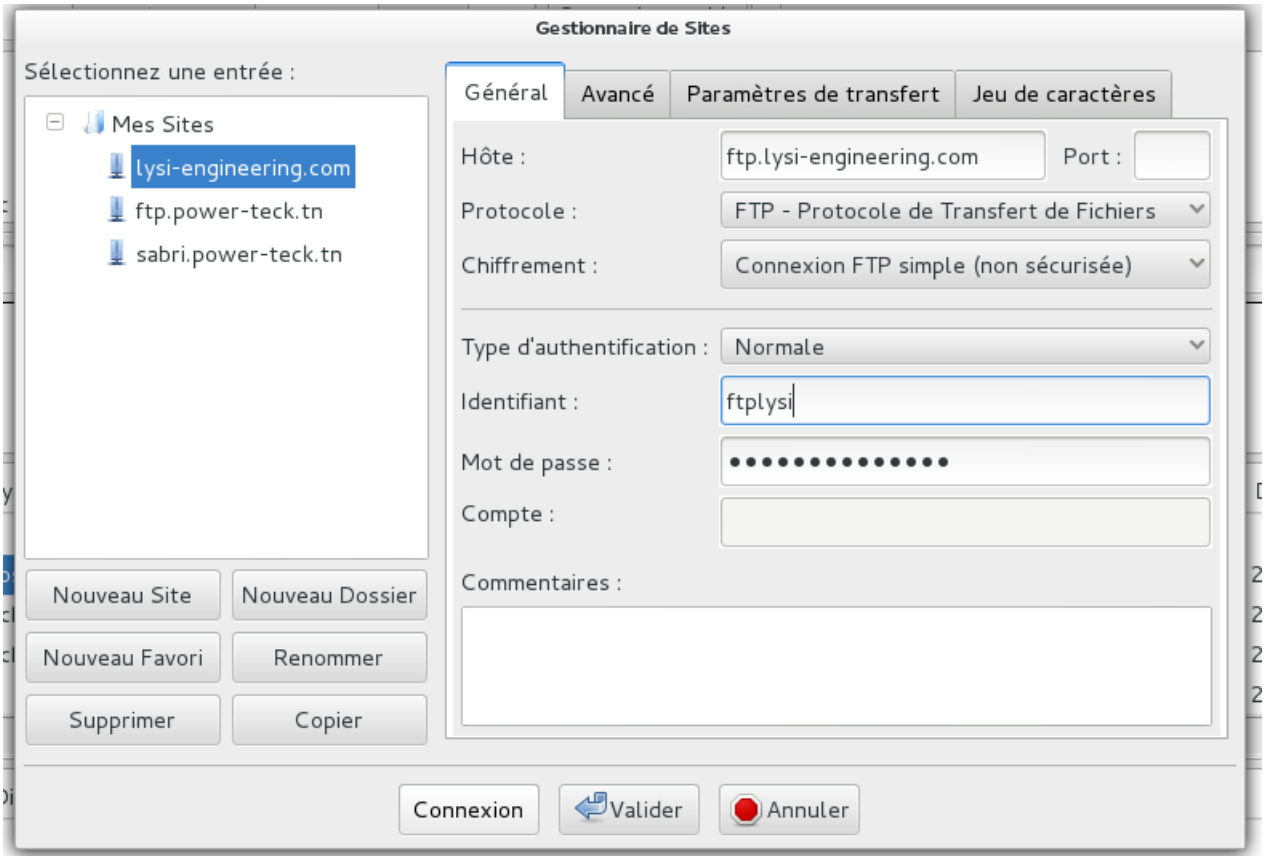

FIGURE IV.4 – Interface d'authentification développeur

Pour s'authentifier le développeur doit remplir les champs de connexion. L'hôte c'est l'adresse du serveur web, le port c'est le port de communication (EX : 8080). Il choisit ensuite le protocole de transfert (FTP, SFTP) et le chiffrement, enfin, il remplie les deux champs à savoir nom d'utilisateur FTP ou SFTP et mot de passe.

5.2 Chargement des fichiers dans le serveur web

Après l'authentification une autre fenêtre s'affiche pour permettre le transfert des fichiers à partir de l'ordinateur personnel au serveur web. Nous pouvons envoyer plusieurs fichiers ayant différents types et tailles en les ajoutant à la file d'attente pour les traiter. Si une ou plusieurs erreurs se produisent lors de l'envoi, le transfert des fichiers ne sarrette pas et l'erreur s'ajoute automatiquement dans la file d'erreur pour la traiter après.

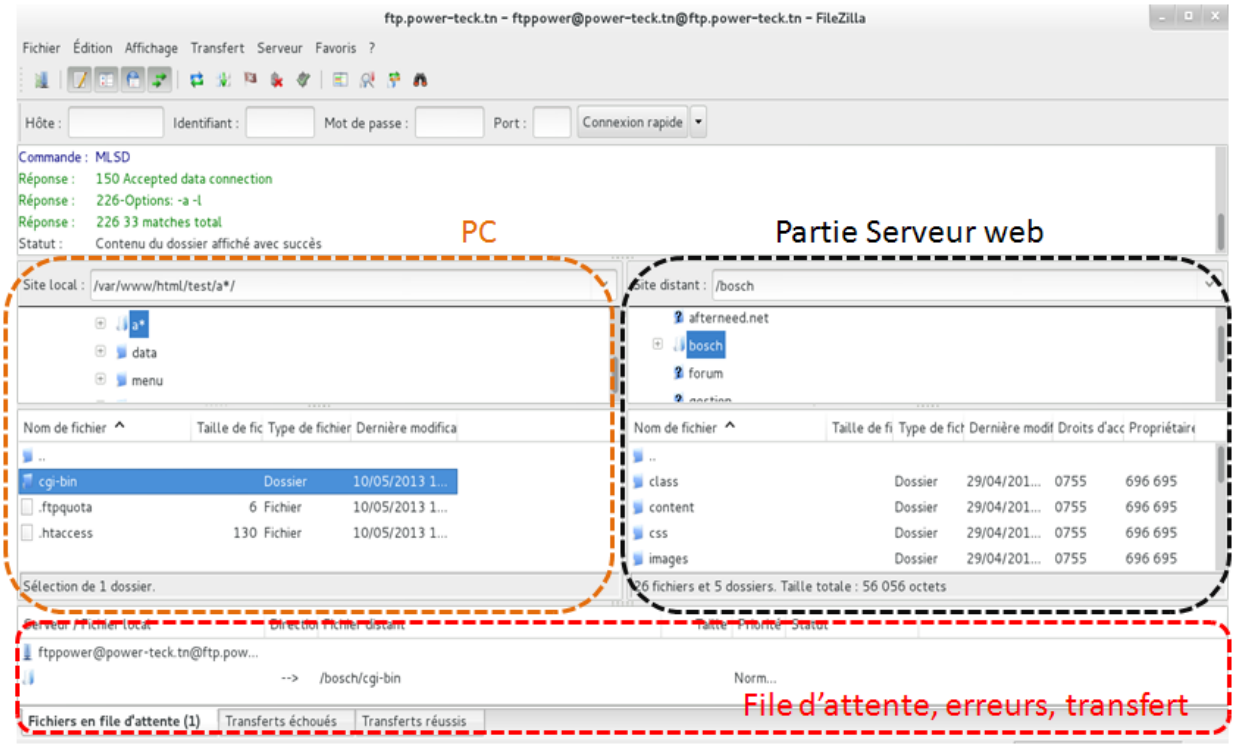
ftp.power-teck.tn - ftppower@power-teck.tn@ftp.power-teck.tn - FileZilla
Fichier Édition Affichage Transfert Serveur Favoris ?
Hôte : Identifiant : Mot de passe : Port : Connexion rapide
Commande : MLSD
Réponse : 150 Accepted data connection
Réponse : 226-Options: -a -l
Réponse : 226 33 matches total
Statut : Contenu du dossier affiché avec succès
PC
Partie Serveur web
Site local : /var/www/html/test/a*/
Site distant : /bosch
a*
data
menu
afterneed.net
bosch
forum
Nom de fichier Taille de fic Type de fichier Dernière modifica
cgi-bin Dossier 10/05/2013 1...
.ftpquota 6 Fichier 10/05/2013 1...
.htaccess 130 Fichier 10/05/2013 1...
Nom de fichier Taille de fi Type de fict Dernière modif Droits d'acc Propriétaire
class Dossier 29/04/201... 0755 696 695
content Dossier 29/04/201... 0755 696 695
css Dossier 29/04/201... 0755 696 695
images Dossier 29/04/201... 0755 696 695
Sélection de 1 dossier.
26 fichiers et 5 dossiers. Taille totale : 56 056 octets
ftppower@power-teck.tn@ftp.pow...
--> /bosch/cgi-bin
Norm...
Fichiers en file d'attente (1)
Transferts échoués
Transferts réussis
File d'attente, erreurs, transfert

FIGURE IV.5 – Interface (Filzilla) de communication avec le serveur web

5.3 Génération d'une nouvelle application

Le système S.O.R se charge de la création d'une nouvelle application après un téléchargement (upload) correcte des fichiers de configuration par le développeur, par la suite, ce dernier crée les administrateurs de l'application générée par défaut.

5.4 Authentification d'utilisateur simple

Afin d'accéder à la plateforme générée par notre S.O.R, l'utilisateur simple doit saisir l'URL de l'application (EX :lysi-engineering.com/bosh) et saisit ensuite son nom d'utilisateur et son mot de passe dans les champs qui s'affiche dans la fenêtre du navigateur utilisée.

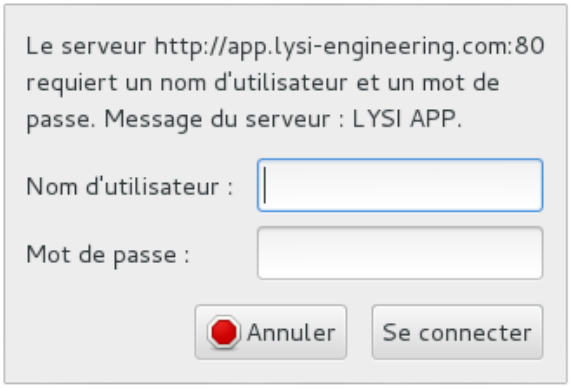
Le serveur http://app.lysi-engineering.com:80 requiert un nom d'utilisateur et un mot de passe. Message du serveur : LYSI APP.
Nom d'utilisateur :
Mot de passe :
Annuler
Se connecter

FIGURE IV.6 – Interface d'authentification des utilisateurs

En cas d'erreur, le message illustré par la figure ci dessous s'affiche.

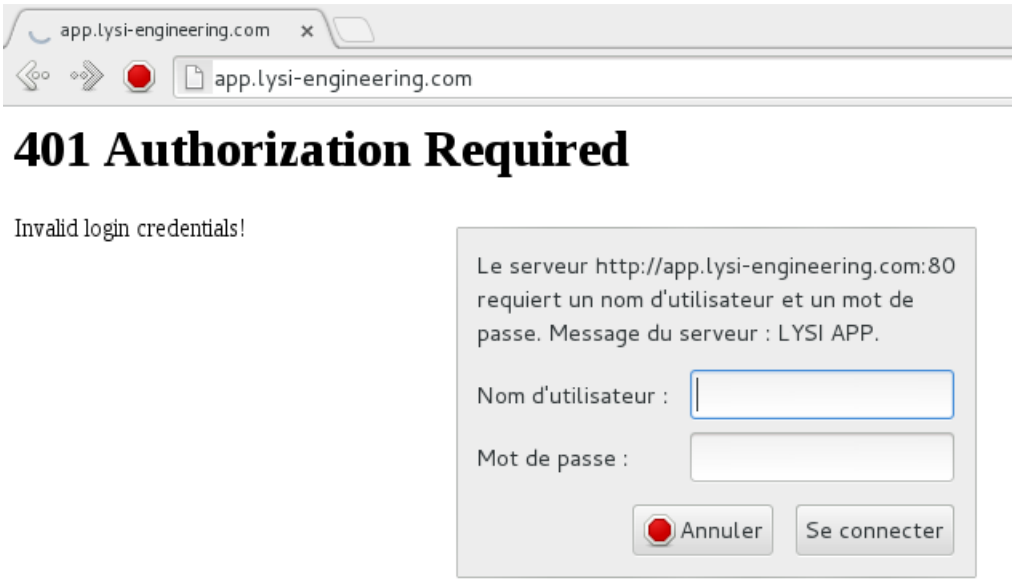
app.lysi-engineering.com
app.lysi-engineering.com
401 Authorization Required
Invalid login credentials!
Le serveur http://app.lysi-engineering.com:80 requiert un nom d'utilisateur et un mot de passe. Message du serveur : LYSI APP.
Nom d'utilisateur :
Mot de passe :
Annuler
Se connecter

FIGURE IV.7 – Interface de message d'erreur d'authentification

5.5 Accès à l'interface du la plateforme

Dans le cas de succès d'authentification, le système redirige l'utilisateur à une interface générée automatiquement lors de l'appel, cette dernière se base sur une structure par défaut pour toutes les applications du S.O.R.

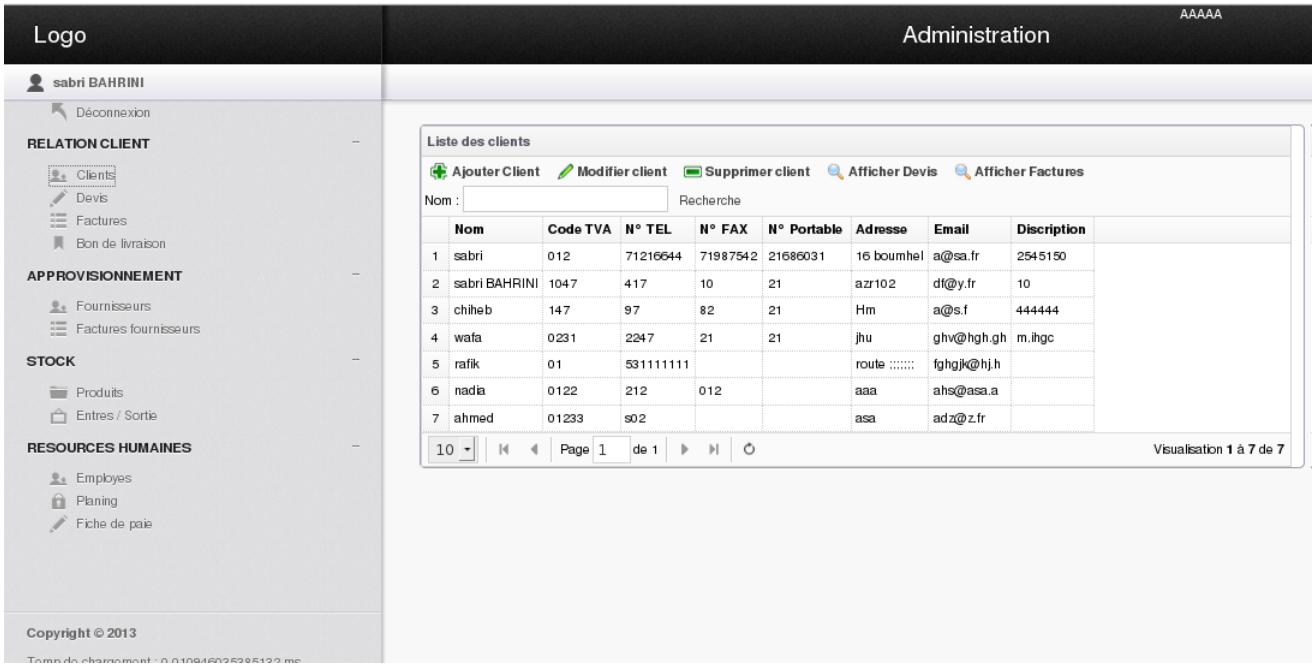

FIGURE IV.8 – Interface de la plateforme générée par S.O.R

6 Définition de système objet relationnel (S.O.R)

La figure ci-dessous montre les différentes couches de notre système (S.O.R).

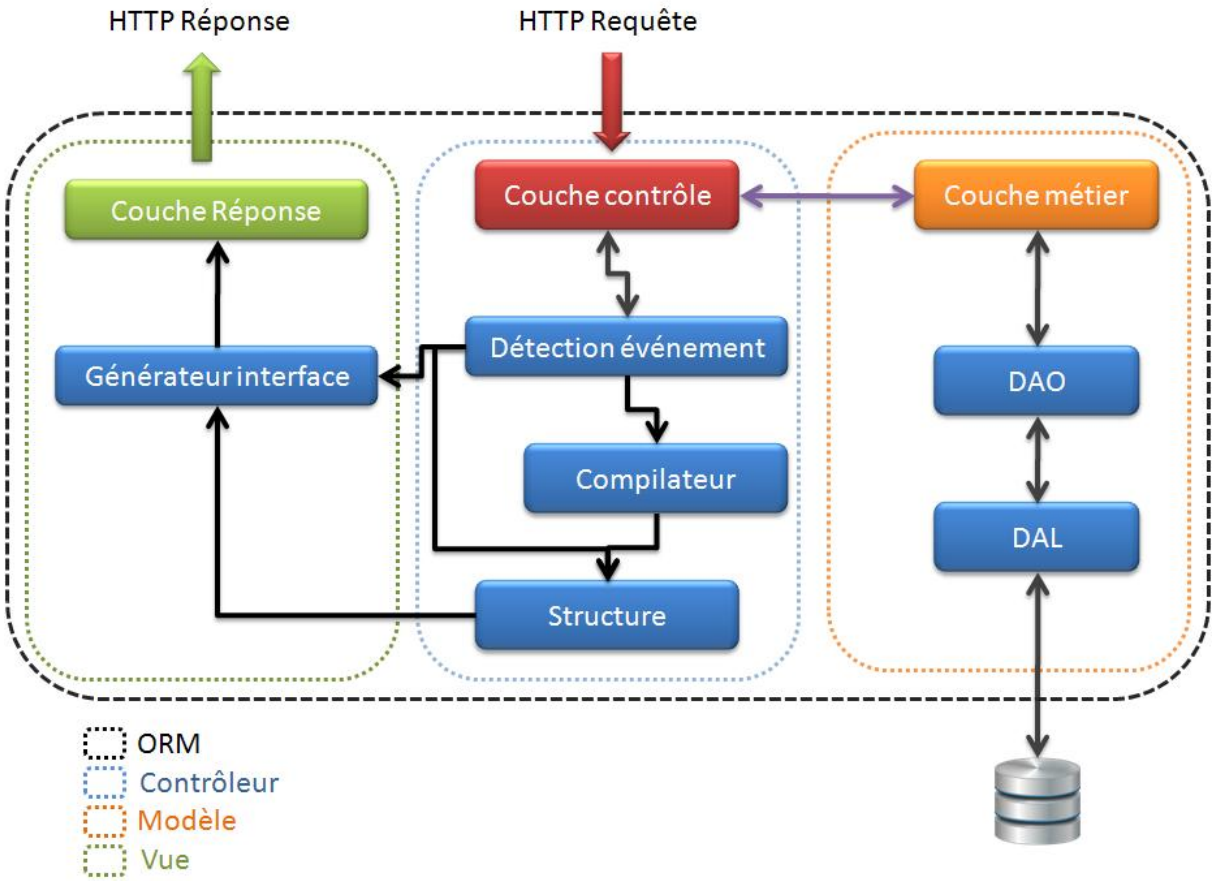

FIGURE IV.9 – Les différentes couches du système objet relationnel (S.O.R)

6.1 Le Modèle S.O.R

Sous sa forme la plus brute, la couche modèle peut être vue comme « données », c'est-à-dire tout ce que nous pouvons lire à partir d'une source, et modifier pour le relire en cas de besoin. Dans une logique de découplage, l'objectif est d'approché le plus possible de la forme brute des données. C'est ce que va faire le modèle en transformant des données brutes en objets structurés, utilisables simplement par la couche inférieure (le contrôleur).

Pour réaliser ce découplage le modèle utilise 3 couches d'abstraction :

- La couche métier : couche de communication et transfert des données avec la couche supérieure (contrôleur).
- La DAL (Data Access Layer) : couche abstraction de données.

– Le DAO (Data Access Object) : objet d'accès aux données.

6.1.1 La DAL (Data Access Layer)

La DAL Permet de s'abstraire du support des données. Pour se faire elle met à disposition des méthodes génériques permettant d'accomplir des actions de maintenances sur les données. Les actions les plus communes sont regroupées sous l'acronyme CRUD (Create Read Update Delete). Basiquement la DAL va donc mettre à disposition des méthodes permettant d'ajouter, mettre à jour, lire, supprimer un enregistrement, et ce quelque soit le support de stockage des données.

La généricité par rapport au stockage est en général matérialisée par un paramètre permettant de spécifier la nature du support (on appelle ça des drivers). Ainsi les méthodes CRUD associées au support de stockage sont utilisées de manière transparente par notre système. Concrètement il n'y a donc en théorie qu'un paramètre à changer pour qu'une application utilisant une DAL puisse changer de support.

6.1.2 Le DAO (Data Access Object)

Le Dao a pour but de transformer les données contenues dans une bases de données en objets et inversement

Pour se faire il va faire correspondre (de manière bijective, ca veut dire que nous pouvons rajouter « et inversement » à la fin de chacun des points suivants) :

- Une table (appelée aussi relation) à une liste d'objets.
- Une ligne d'une table (appelée aussi tuple) à un objet.
- Un champs de base de données à un attribut d'objet.
- Une valeur d'un champs à une valeur d'attribut d'un objet.

Techniquement la DAO interroge le SGBD via la DAL sur la structure des tables afin de maintenir la correspondance entre les champs des tables de la base de données et les attributs des objets.

6.1.3 La couche métier de S.O.R

Le but des couches précédentes est de facilité les tache de la couche métier. Si cette dernier hérite des méthodes de la DAO et des classes de l'ORM, nous aurons des objets

présentant toutes les méthodes pour les gérer en base.

6.2 Le contrôleur S.O.R

Le contrôleur de notre S.O.R est différent que les autres contrôleurs, il contient des couches bien structurées pour assurer la rapidité et la performance d'échange des données entre les différent espaces et couches.

En effet, les contrôleurs font appel à la séquence CRUD du modèle, par l'intermédiaire d'une relation d'héritage simple, afin de permettre de personnaliser le traitement par contrôleur en cas de besoin.

Les attributs de ces classes sont générées à partir de la compilation de la structure XML qui assure tout le MAPPING.

6.3 La vue S.O.R

La vue du système objet relationnel ne contient que la couche de réponse et la couche de génération des interfaces à partir de la structure compilée par le contrôleur.

Il s'agit de fournir des structures de données capables de générer toute sorte de composants graphiques nécessaire à l'interface Homme machine, sous format HTML.

Les types de composants ci-dessous qui constituent notre interface. Chacun d'entre eux correspond à une classe qui contient les attributs et les méthodes nécessaires afin de générer le code en question.

L'affichage varie selon le contrôleur invoqué.

6.3.1 Menu

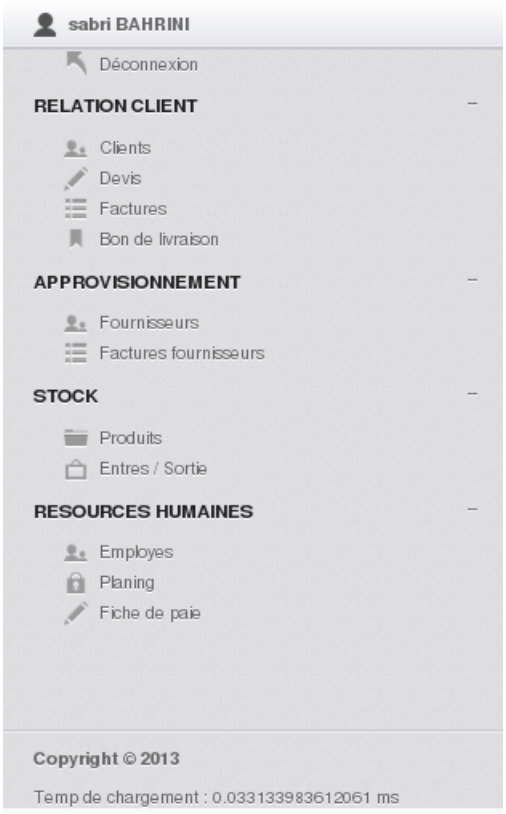

FIGURE IV.10 – Exemple d'un menu final d'une application gérer par S.O.R

Le menu principal est défini dans de la couche DATA. Il est construit à partir d'une structure de balises dans un fichier de configuration propre à lui.

L'affichage des entrées dépend du niveau d'accès de l'utilisateur connecté, et à chacune d'entre elle correspond un contrôleur bien déterminé.

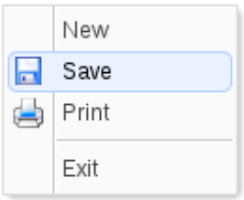

FIGURE IV.11 – Exemple d'un menu secondaire

Ceci est un menu contextuel, qui permet d'appliquer des actions spécifiques sur des objets en cas de besoin.

6.3.2 Vue

Simulé du forme de DATAGRID, l'objet « vue » prend en paramètre un contrôleur qui lui fournit des colonnes à extraire, le nombre de lignes à afficher par page, des actions qui peuvent être lancées sur les objets de la vue.
Il permet aussi de fournir le formulaire de recherche afin de restreindre les requêtes de sélection.

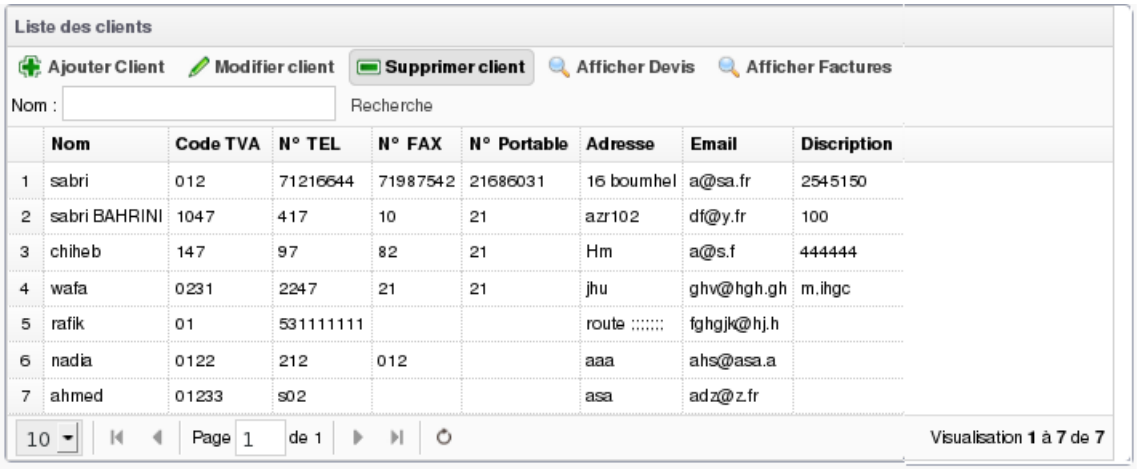
Liste des clients
Ajouter Client
Modifier client
Supprimer client
Afficher Devis
Afficher Factures
Nom :
Recherche
Nom | Code TVA | N° TEL | N° FAX | N° Portable | Adresse | Email | Discription
1 | sabri | 012 | 71216644 | 71987542 | 21686031 | 16 boumhel | a@sa.fr | 2545150
2 | sabri BAHRINI | 1047 | 417 | 10 | 21 | azr102 | df@y.fr | 100
3 | chiheb | 147 | 97 | 82 | 21 | Hm | a@s.f | 444444
4 | wafa | 0231 | 2247 | 21 | 21 | jhu | ghv@hgh.gh | m.ihgc
5 | rafik | 01 | 531111111 | | | route ;;;;;; | fghgjk@hj.h |
6 | nadia | 0122 | 212 | 012 | | aaa | ahs@asa.a |
7 | ahmed | 01233 | s02 | | | asa | adz@z.fr |
10
Page 1 de 1
Visualisation 1 à 7 de 7

FIGURE IV.12 – Exemple d'une DataGrid

6.3.3 Formulaire

L'objet « Formulaire » a un mécanisme quasi-identique à la vue, il interroge le contrôleur en question afin d'extraire la liste des champs à afficher avec leurs types et leurs modes d'affichage, et permet à l'utilisateur de déclencher les évènements d'ajout ou de modification de données.

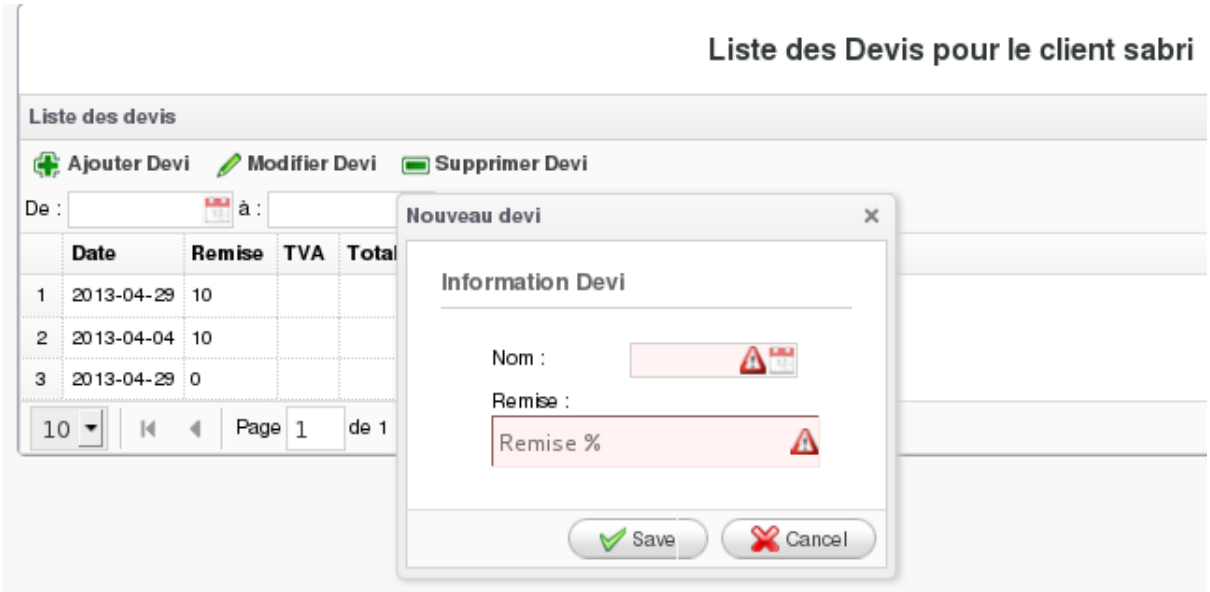

FIGURE IV.13 – Exemple d'un formulaire

6.3.4 Dialogue

Les Dialogues représentent les méthodes affectées au contrôleur associé à la vue ou au formulaire. Ils permettent de lancer un traitement spécifique sur un ou plusieurs objets.

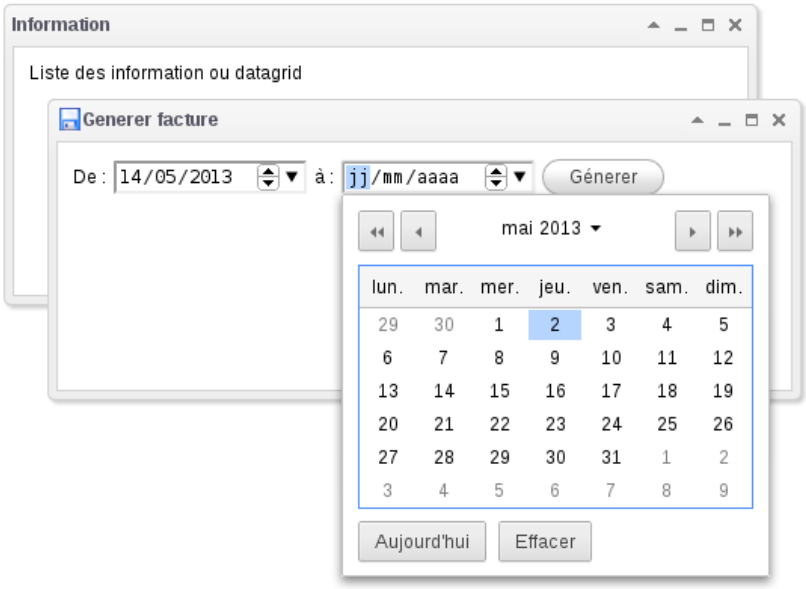
Information
Liste des information ou datagrid
Generer facture
De : 14/05/2013
à : jj/mm/aaaa
Générer
mai 2013
lun. mar. mer. jeu. ven. sam. dim.
29 30 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31 1 2
3 4 5 6 7 8 9
Aujourd'hui
Effacer

FIGURE IV.14 – Exemple de dialogue

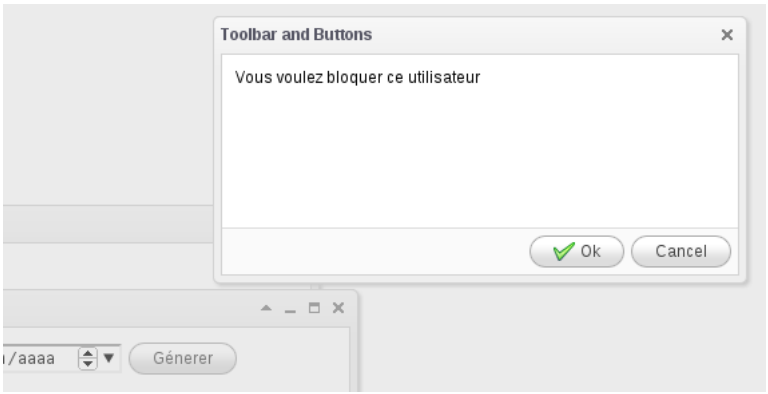

FIGURE IV.15 – Exemple de dialogue message

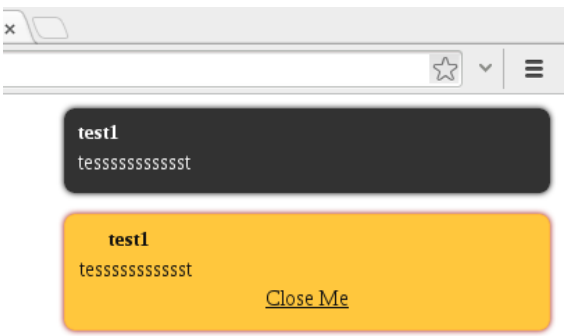

FIGURE IV.16 – Exemple de message de notification

7 Définition du compilateur de S.O.R

Pour assurer la cohérence de notre système d'information, nous avons opté pour un mappeur à la fois extensible et facile à compiler, afin de simplifier la réalisation, éviter la dépendance à un langage de programmation bien déterminé et garantir une éventuelle évolution vers d'autres technologies.

Ceci explique le choix de XML pour réaliser le MAPPING entre le modèle objet et le modèle relationnel dans notre plateforme.
La plupart des langages permettent de parcourir les fichiers XML (PARSER XML), mais ce qui est plus important c'est qu'il suffit de définir un DTD pour fixer la séquence logique à respecter, évitant ainsi la complexité des compilateurs classiques.

Une fois la structure des documents de configuration est conforme, il ne reste plus qu'à créer un PARSEUR qui, à partir d'une séquence attendue de balise, balises génére la structure de l'application, à savoir les attributs des contrôleurs pour le modèle orienté objet, ainsi que tous les objets de la base de données pour le modèle relationnel, et finalement de générer l'interface homme machine.

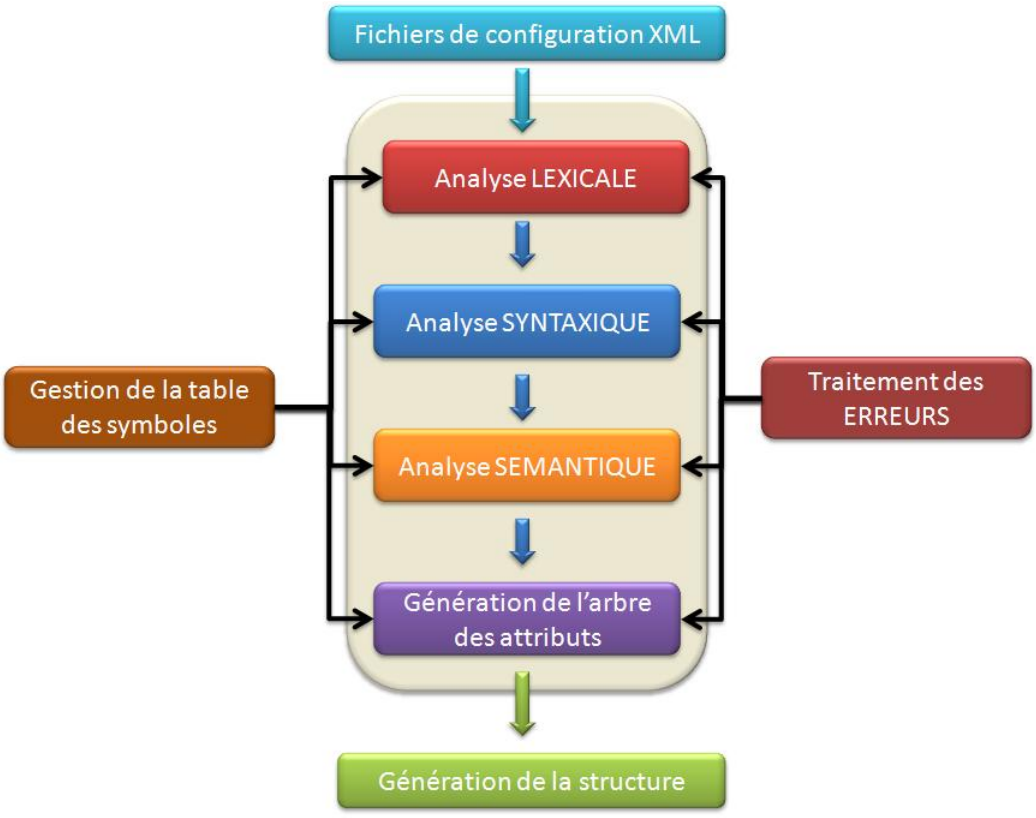

FIGURE IV.17 – Compilateur S.O.R

Conclusion

La réalisation de projet telle que conçue dans ce chapitre a été basée sur des nouvelles technologies qui sont toutes libres. L'implémentation de l'application a été favorisée par le choix judicieux de l'environnement de développement, qui nous a permis d'ailleurs une mise en œuvre d'un système dont nous avons décrit les grandes lignes.

CONCLUSION ET PERSPECTIVES

Durant la période de ce stage, notre mission était de concevoir et d'implémenter une plateforme web permettant la création d'un mapping objet relationnel au sein de la société LYSI ENGENEERING, afin de faciliter la gestion des SGBD aux concepteurs et aux développeurs de l'application.

En se basant sur la formation que nous avons acquise durant notre cursus universitaire ainsi que les nouvelles notions que nous avons pu assimiler au sein de l'entreprise dans laquelle nous avons effectuer notre stage, nous sommes arrivés à réaliser les objectifs que nous avons fixés en respectant la durée prévu pour la mise en place de notre plateforme.

Ce stage de fin d'études nous a permis de mettre en œuvre nos savoirs et nos connaissances théoriques, et de les enrichir par la pratique, ainsi que de maitriser un ensemble de nouvelles technologies. Cette expérience a été une opportunité pour apprendre en avantage sur les systèmes de gestion de base de données, de s'approfondir dans le développement web dans sa globalité, assimiler ses concepts et manipuler ses outils.

Nous avons également appris à être encore plus autonomes et responsables afin de bien s'intégrer dans le milieu professionnel.
Ce projet nous a été une source de bénéfices valorisants tant au niveau technique qu'au niveau professionnel et relationnel.

Nous tenons à souligner que la solution développée reste extensible à tout type d'amélioration et d'ajout de nouveaux modules tels que les styles d'affichage, la sécurité des données, voir même les objets pris en considération avec notre système objet relationnel.

En guise de conclusion, on dirait que notre projet se suite dans une voie en pleine expansion puisque le concept de l'orienté objet et les systèmes de gestion de base de données sont en évolution croissante.

Bibliographie

[1] Apache http server. http://fr.wikipedia.org/wiki/Apache_HTTP_Server.

[2] Doctrine. http://www.doctrine-project.org/. 2010.

[3] Html5. https://fr.wikipedia.org/wiki/HTML5.

[4] Javascript object notation. http://fr.wikipedia.org/wiki/JavaScript_Object_Notation.

[5] jquery. http://fr.wikipedia.org/wiki/JQuery.

[6] Modèle-vue-contrôleur. http://fr.wikipedia.org/wiki/Mod%C3%A8le-Vue-Contr%C3%B4leur.

[7] Modèle-vue-contrôleur. http://www.mysql.fr/products/workbench/.

[8] Modèle-vue-contrôleur. http://fr.wikipedia.org/wiki/FileZilla.

[9] Mysql. http://fr.wikipedia.org/wiki/MySQL.

[10] Php : Hypertext preprocessor. https://fr.wikipedia.org/wiki/PHP.

[11] Smart, easy object persistence. http://propelorm.org/.

[12] Unified modeling language. http://fr.wikipedia.org/wiki/Unified_Modeling_Language.